M T S

The Michigan Terminal System

Volume 2: Public File Descriptions

January 1987

Updated August 1988 (Update 1)
Updated December 1988 (Update 2)
Updated May 1990 (Update 3)

The University of Michigan Computing Center
Ann Arbor, Michigan

```
****************************************************
*                                                  *
*       This obsoletes the April 1982 edition.     *
*                                                  *
****************************************************
```

DISCLAIMER

The MTS Manual is intended to represent the current state of the Michigan Terminal System (MTS), but because the system is constantly being developed, extended, and refined, sections of this volume will become obsolete. The user should refer to the U-M Computing News and other ITD documentation for the latest information about changes to MTS.

January 1987   Page Revised May 1990

PREFACE

   The software developed by the Information Technology Division (ITD)
technical staff for the operation of the high-speed processor computer
can be described as a multiprogramming supervisor that handles a number
of resident, reentrant programs. Among them is a large subsystem,
called MTS (Michigan Terminal System), for command interpretation,
execution control, file management, and accounting maintenance.   Most
users interact with the computer's resources through MTS.

   The MTS Volumes are a series of manuals that describe in detail the
facilities provided by the Michigan Terminal System.   Administrative
policies ITD and the physical facilities provided are described in other
publications.

   The MTS Volumes now in print are listed below. The date indicates
the most recent edition of each volume; however, since volumes are
periodically updated, users should check the file *CCPUBLICATIONS, or
watch for announcements in the U-M Computing News, to ensure that their
MTS Volumes are fully up to date.

   The numerical order of the volumes does not necessarily reflect the
chronological order of their appearance; however, in general, the higher

the number, the more specialized the volume. Volume 1, for example,
introduces the user to MTS and describes in general the MTS operating
system, while Volume 10 deals exclusively with BASIC.

   The attempt to make each volume complete in itself and reasonably
independent of others in the series naturally results in a certain
amount of repetition. Public file descriptions, for example, may appear
in more than one volume. However, this arrangement permits the user to
buy only those volumes that serve his or her immediate needs.

                                Richard A. Salisbury

                                General Editor

January 1987

PREFACE TO REVISED VOLUME 2

   The January 1987 revision reflects the changes that have been made to MTS since April 1982. Some of these changes were described in Updates 1 through 6 that were issued from February 1983 through May 1986 and which are incorporated in this revision. However, several changes have occurred since May 1986 and, hence, it was felt that a complete revision of this volume was in order. The revision bars have been deleted and the pages have been renumbered to facilitate the future issuing of updates.

Contents

January 1987

<div align="center">CHANGES IN PUBLIC FILE DESCRIPTIONS</div>

   The following public file descriptions have been added to this volume since Update 6 (May 1986) to the March 1982 edition was issued.

    *CMS
    *CMSTAPELOAD, *CMSTAPESCAN
    *COMPUCALENDAR
    *MACBINHEX
    *MACPAINT4045
    *MACWRITE9700
    *SLIDEQ

   The following public file descriptions have been deleted from this volume. Some of these public files still exist in the system, but the Computing Center makes no guarantee as to how long they will exist. Documentation for some of these programs can be found in the file DOC:VOLUME.2X.

    *CCMEMOS   Information combined into *CCPUBLICATIONS
    *CCQUICKNOTES   Information combined into *CCPUBLICATIONS
    *COBOLU   Program seldom used (replaced by *COBOLVS)
    *COINFLIP   Program seldom used
    *DOWNDATE   Program seldom used
    *FAKEOS   Program seldom used (replaced by *VSS)
    *FILEDUMP   Program seldom used
    *FILESCAN   Program seldom used
    *FORMAT   Described in MTS Volume 15, FORMAT and TEXT360
    *FORTRANG   Described in MTS Volume 6, FORTRAN in MTS
    *FORTRANH   Described in MTS Volume 6, FORTRAN in MTS
    *GENDOCDBMS   Described in CCMemo 461
    *KEYPUNCHES   Information combined into *WORKSTATIONS
    *LISP   Program seldom used (replaced by *UTILISP)
    *NIM   Program seldom used
    *PIL   Program seldom used
    *PLOT   Program seldom used
    *PL360   Program seldom used
    *PRINT   Described in MTS Volume 15, FORMAT and TEXT360
    *SIDEDATE   Program seldom used
    *SLIP   Library seldom used
    *STATUS   Program seldom used (replaced by $ACCOUNTING)
    *TERMINALS   Information combined into *WORKSTATIONS
    *TEXT360   Described in MTS Volume 15, FORMAT and TEXT360
    *UPDATE   Program seldom used

INDEX OF PUBLIC FILES

   The following table lists all of the public files in the system that
are of general use to users.   Other public files exist within the
system, but they are associated with internal system operations and are
not listed.   The list below is also contained in the public file
*CCSOFTWARE.

   For each public file, a subject category and a documentation
reference is given.   For public files that contain obsolete or seldom
used programs, the reference is DOC:VOLUME.2X.   This volume is main
tained on-line in the file DOC:VOLUME.2X and may be copied by issuing
the command

| $COPY DOC:VOLUME.2X *PRINT*

Public File	Subject	Reference
*ALGOL	OS/360 ALGOL	DOC:VOLUME.2X
*ALGOLLIB	OS/360 ALGOL	DOC:VOLUME.2X
*ALGOLW	ALGOL W	MTS Volume 2; MTS Volume 16
*ALGOLWLIB	ALGOL W	MTS Volume 16
*AMENDS	File Comparison	DOC:VOLUME.2X
*ANALYSIS	System Statistics	MTS Volume 2; CCMemo 393
*ANALYZER	XPL	DOC:VOLUME.2X
*APC	File Comparison	MTS Volume 2
*APL	VS APL	MTS Volume 2; CCMemo 435
*APLEDCL	General Motors APL	DOC:VOLUME.2X; CCMemo 236
*APLFONTCONVERT	VS APL	MTS Volume 2
*APLGM	General Motors APL	DOC:VOLUME.2X; CCMemo 236
*APLLIB	General Motors APL	DOC:VOLUME.2X; CCMemo 236
*APLTRAN	General Motors APL	None
*ARC	File Archiving	MTS Volume 2, CCMemo 484
*ASA	Carriage-Control Conv.	DOC:VOLUME.2X
*ASCEBCD	ASCII-EBCDIC Conversion	MTS Volume 2
*ASMBLR	Same as *ASMG	
*ASMG	360/370 Assembler	DOC:VOLUME.2X
*ASMGSYSMAC	360/370 Assembler	DOC:VOLUME.2X
*ASMH	360/370 Assembler	MTS Volume 2; MTS Volume 14
*ASMT	360/370 Assembler	DOC:VOLUME.2X
*ASMTIDY	360/370 Assembler	MTS Volume 2; MTS Volume 14
*ASMTSYSMAC	360/370 Assembler	DOC:VOLUME.2X
*ASSIST	360/370 Assembler	MTS Volume 2; MTS Volume 14
*ASSISTMAC	360/370 Assembler	MTS Volume 14
*AUTOSTART	Batch Processing	MTS Volume 2
*BASIC	Same as *BASICUM	
*BASICUM	U of M BASIC	MTS Volume 2; MTS Volume 10
*BCDEBCD	BCD-EBCDIC Conversion	DOC:VOLUME.2X

```
      *BIBTEX            Text Processing (TEX)     MTS Volume 2
      *BIGTEX            Text processing (TEX)     None
      *BRAILLE           Braille Output            None
      *CAMPUSZIPS        Campus Zip Listing        None
      *CBELL             C Programming Language    MTS Volume 2
|     *CBELLASMMAC       C Programming Language    None
      *CBELLINCLUDE      C Programming Language    *CBELL, MTS Volume 2
      *CBELLLIB          C Programming Language    *CBELL, MTS Volume 2
      *CCBIBLIOGRAPHY    Documentation Info.       MTS Volume 2
      *CCDELIVERY        Output Delivery Info.     MTS Volume 2
      *CCHOURS           Building Information      MTS Volume 2
      *CCMEMOS           Documentation Info.       None
      *CCNEWSDEADLINES   Documentation Info.       None
      *CCPHONES          Telephone Information     MTS Volume 2
      *CCPUBLICATIONS    Documentation Info.       MTS Volume 2
      *CCQUEUE           Plot System               MTS Volume 2; MTS Volume 11
      *CCRESHOURS        Residence Hall Info.      None
      *CCSOFTWARE        Software Information       MTS Volume 2
      *CDUPDATE          File Updating             MTS Volume 2; CCMemo 434
      *CKID              Signon ID Security        MTS Volume 2
      *CKSIG             Signon ID Security        MTS Volume 2
      *CLPARSEGEN        Syntactic Parser          MTS Volume 2; *CLPARSER.W
      *CLPARSER          Syntactic Parser          MTS Volume 2; *CLPARSER.W
      *CLPARSELIB        Syntactic Parser          *CLPARSER.W
      *CMDMACLIB         MTS Command Macros        MTS Volume 2
      *CMS               IBM VS/CMS simulator      MTS Volume 2; CCMemo 448
      *CMSFILETYPEMAP    IBM VS/CMS Simulator      MTS Volume 2
      *CMSTAPELOAD       Magnetic Tapes            MTS Volume 2; MTS Volume 19
      *CMSTAPESCAN       Magnetic Tapes            MTS Volume 2; MTS Volume 19
      *CNFGINFODSECT     Subroutine Dsect          CNFGINFO, MTS Volume 3
      *COBLIB            COBOL                     *COBOLU, DOC:VOLUME.2X
      *COBLINKEDIT       COBOL VS                  CCMemo 439
      *COBOLU            COBOL                     DOC:VOLUME.2X
      *COBOLVS           COBOL VS                  MTS Volume 2; CCMemo 439
      *COBOLVSRUN        COBOL VS                  CCMemo 439
      *COINFLIP          Demonstration             DOC:VOLUME.2X
      *COLLATE           Record Collating          MTS Volume 2
      *COMBINE           Record Combination        MTS Volume 2
      *COMPARE           File Comparison           MTS Volume 2
      *COMPUCALENDAR     Information               MTS Volume 2
      *CONDITIONSOFUSE   Information               MTS Volume 2
      *CONSULTINGHOURS   Counseling Information    MTS Volume 2
      *CONSULTINGSCHED   Counseling Information    MTS Volume 2
      *CONVSNOBOL        SNOBOL4                   MTS Volume 9
      *COUNSELINGHOURS   Same as *CONSULTINGHOURS
|     *CRISPINFO         U-M Course Info.          None
      *CRCBOOKLIST       CRC reference books       MTS Volume 2
      *CRCSOFTWARE       CRC evaluation software   MTS Volume 2
      *CSMP              CSMP                      MTS Volume 2
      *CSMPEXEC          CSMP                      *CSMPTRAN, MTS Volume 2
      *CSMPLIB           CSMP                      *CSMPTRAN, MTS Volume 2
      *CSMPTRAN          CSMP                      MTS Volume 2
|     *C87               C Programming Lang.       MTS Volume 2, R1032
```

```
*C87INCLUDE       C Programming Lang.     *C87, MTS Volume 2
*C87LIB           C Programming Lang.     *C87, MTS Volume 2
*DAVE             Data-flow Analyzer      MTS Volume 2; CCMemo 394
*DAVE.GENLIB      Data-flow Analyzer      CCMemo 402
*DCALC            Numerical               DOC:VOLUME.2X
*DECODE           File Compression        DOC:VOLUME.2X
*DEDIT            Distribution Editor      MTS Volume 2
*DELIVERY         Same as *CCDELIVERY     MTS Volume 2
*DICT             TEXT360                 MTS Volume 15
*DICTUPD          TEXT360                 MTS Volume 15
*DITTO            Ditto Master Program    MTS Volume 2
*DI3000.IG        DI-3000 Graphics Library MTS Volume 2; CCMemo 463
*DI3000.PRINTER   DI-3000 Graphics Library MTS Volume 2; CCMemo 463
*DI3000.TX4010    DI-3000 Graphics Library MTS Volume 2; CCMemo 463
*DI3000.TX4113    DI-3000 Graphics Library MTS Volume 2; CCMemo 463
*DOCINFO          Online Documentation    MTS Volume 2
*DOWNDATE         File Comparison         DOC:VOLUME.2X
*DVIAPS           Text Processing (TEX)   MTS Volume 2; CCMemo 810
*DVITYPE          Text Processing (TEX)   CCMemo 811
*DVIXER           Text Processing (TEX)   MTS Volume 2; CCMemo 810
*EBCDASC          EBCDIC-ASCII Conversion  MTS Volume 2
*EBCDBCD          EBCDIC-BCD Conversion   DOC:VOLUME.2X
*ENCODE           File Compression        DOC:VOLUME.2X
*ENCRYPT          File Security           MTS Volume 2
*ENDJUNK          Loader Records          MTS Volume 2; MTS Volume 5
*EXCOM            Extended-XPL            *EXPL, MTS Volume 2; CCMemo
                                          416
*EXPL             Extended-XPL            MTS Volume 2; CCMemo 416
*EXPLBNF          Extended-XPL            CCMemo 416
*EXPLIB           Extended-XPL            *EXPL, MTS Volume 2; CCMemo
                                          416
*EXSKELETON       XPL                     *EXPL, MTS Volume 2; CCMemo
                                          416
*FAKEOS           OS/360 Environment      DOC:VOLUME.2X
*FAKEOSCREBLDL    OS/360 Environment      *FAKEOS, DOC:VOLUME.2X
*FAKEOSTRACE      OS/360 Environment      *FAKEOS, DOC:VOLUME.2X
*FILEDUMP         File Dumping            DOC:VOLUME.2X
*FILESCAN         File Scanning           DOC:VOLUME.2X
*FLOPDSECT        Floppy Disk Sense Data  None
*FLOWCHART        Flowcharting            MTS Volume 2
*FOREVER          Unlimited output        None
*FORMAT           FORMAT                  MTS Volume 15
*FORTRANG         FORTRAN                 MTS Volume 6
*FORTRANH         FORTRAN                 MTS Volume 6
*FORTRANVS        FORTRAN                 MTS Volume 2; MTS Volume 6
*FORUM            User Conference         MTS Volume 2; MTS Volume 23
*FS               File Saving             MTS Volume 2; MTS Voume 19
*FSAVE            File Saving             DOC:VOLUME.2X
*FSCON            File Saving             DOC:VOLUME.2X
*FTN              FORTRAN                 MTS Volume 2; MTS Volume 6
*FTNGTEST         FORTRAN                 MTS Volume 6
*FTNTIDY          FORTRAN                 MTS Volume 2; MTS Volume 6
*FTNTOPL1         FORTRAN-PL/I Conversion  MTS Volume 6
```

```
  *GASP            GASP                      MTS Volume 2; CCMemo 246
  *GASPCOM         GASP                      CCMemo 246
  *GASPIV          GASP                      CCMemo 246
  *GASPIVCOM       GASP                      CCMemo 246
  *GDINFODSECT     Subroutine Dsect          GDINFO, MTS Volume 3
  *GENLIB          Library Generator         DOC:VOLUME.2X
  *GFINFODSECT     Subroutine Dsect          GFINFO, MTS Volume 3
  *GOM             MAD Programming Language  None
  *GPSS            GPSS                      DOC:VOLUME.2X
  *GPSSERR         GPSS                      DOC:VOLUME.2X
| *GPSSH           GPSS/H                    None
| *GPSSH2          GPSS/H Version 2          MTS Volume 2
  *GPSSLIB         GPSS                      DOC:VOLUME.2X
  *HPGLPLOT        Remote Plotter Support    MTS Volume 2; CCMemo 466
  *HYPHDICT        TEXT360                   MTS Volume 15
  *ICON            ICON Prog. Language       MTS Volume 2
  *ICONLIB         ICON Prog. Language       MTS Volume 2
  *IEBUPDAT        Data Set Utility          CCMemo 449
  *IEHMOVE         Data Set Utility          CCMemo 449
  *IF              Same as *IF66             MTS Volume 2; MTS Volume 6
  *IF66            FORTRAN                   MTS Volume 2; MTS Volume 6
  *IF77            FORTRAN                   MTS Volume 2; MTS Volume 6
  *IG              Integrated Graphics       MTS Volume 2; MTS Volume 17
  *IG.AJ830        Integrated Graphics       MTS Volume 17
  *IG.AMIGO        Integrated Graphics       MTS Volume 17
  *IG.ARU          Integrated Graphics       MTS Volume 17
  *IG.CCMP         Integrated Graphics       MTS Volume 17
  *IG.CK400        Integrated Graphics       MTS Volume 17
  *IG.CRT          Integrated Graphics       MTS Volume 17
  *IG.DDRXF        Integrated Graphics       MTS Volume 17
  *IG.DEBUG        Integrated Graphics       MTS Volume 17
  *IG.DTC300       Integrated Graphics       MTS Volume 17
  *IG.DTC302       Integrated Graphics       MTS Volume 17
  *IG.FPS          Integrated Graphics       MTS Volume 17
  *IG.GT40         Integrated Graphics       MTS Volume 17
  *IG.HIDP11       Integrated Graphics       MTS Volume 17
  *IG.HPGL         Integrated Graphics       MTS Volume 17
  *IG.HP2648A      Integrated Graphics       MTS Volume 17
  *IG.HP7203       Integrated Graphics       MTS Volume 17
  *IG.HP7221       Integrated Graphics       MTS Volume 17
  *IG.HP7470       Integrated Graphics       MTS Volume 17
  *IG.HUGHES       Integrated Graphics       MTS Volume 17
  *IG.MEGATEK      Integrated Graphics       MTS Volume 17
  *IG.MUGRAF       Integrated Graphics       MTS Volume 17
  *IG.MX12000      Integrated Graphics       MTS Volume 17
  *IG.PEP          Integrated Graphics       MTS Volume 17
  *IG.PICK         Integrated Graphics       MTS Volume 17
  *IG.PRINTER      Integrated Graphics       MTS Volume 17
  *IG.PS           Integrated Graphics       MTS Volume 17
  *IG.QUMES5       Integrated Graphics       MTS Volume 17
  *IG.SAVE         Integrated Graphics       MTS Volume 17
  *IG.SCODL        Integrated Graphics       MTS Volume 17
  *IG.SHUFFLE      Integrated Graphics       MTS Volume 17
```

```
*IG.TD4000         Integrated Graphics    MTS Volume 17
*IG.TX4002         Integrated Graphics    MTS Volume 17
*IG.TX4006         Integrated Graphics    MTS Volume 17
*IG.TX4010         Integrated Graphics    MTS Volume 17
*IG.TX4014         Integrated Graphics    MTS Volume 17
*IG.TX4025         Integrated Graphics    MTS Volume 17
*IG.TX4025GIN      Integrated Graphics    MTS Volume 17
*IG.TX4027         Integrated Graphics    MTS Volume 17
*IG.TX4105         Integrated Graphics    MTS Volume 17
*IG.TX4113         Integrated Graphics    MTS Volume 17
*IG.TX4662         Integrated Graphics    MTS Volume 17
*IG.TX4663         Integrated Graphics    MTS Volume 17
*IG.XX1620         Integrated Graphics    MTS Volume 17
*IG.3270           Integrated Graphics    MTS Volume 17
*IG.339            Integrated Graphics    MTS Volume 17
*INDEX             Index Program          MTS Volume 15
*I8080ASR          Intel 8080 Assembler   CCMemo 422
*KDFLIB            OS/360 ALGOL           DOC:VOLUME.2X
*KERMIT            Network File Transfer  MTS Volume 2; CCMemo 473
*KEYPUNCHES        Keypunch Information   None
*LABEL             Magnetic Tapes         MTS Volume 2
*LABELS            Page Printer Output    MTS Volume 2; MTS Volume 19
*LABELSNIFF        Magnetic Tapes         MTS Volume 2; MTS Volume 19
*LALRGEN           Grammar Parser         MTS Volume 2
*LATEX             Text Processing (TEX)  MTS Volume 2; CCMemo 815
*LCS               Loader Records         MTS Volume 2; MTS Volume 5
*LIBRARY           System Library         MTS Volume 2
*LINKEDIT          Linkage Editor         MTS Volume 2; MTS Volume 5
*LINK11            PDP-11 Linkage Editor  MTS Volume 2; CCMemo 286
*LINPG             Numerical              DOC:VOLUME.2X
*LISP              LISP                   MTS Volume 8
*LISPLIB           LISP                   MTS Volume 8
*LOADINFODSECT     Subroutine Dsect       LOADINFO, MTS Volume 3
*LOCALPLOT         Remote Plotter Support MTS Volume 2; CCMemo 465
*LOCALPRINT        Remote Printer Support MTS Volume 2
*LOGPURGE          Log-File Processing    MTS Volume 2
*MACBINHEX         Microcomputers         MTS Volume 2
*MACGEN            Macro Library Generator DOC:VOLUME.2X
*MACPAINT4045      Microcomputers         MTS Volume 2
*MACUTIL           Macro Library Generator MTS Volume 2; MTS Volume 14
*MACWRITE9700      Microcomputers         MTS Volume 2
*MADTOPL1          7090 MAD-PL/I Conversion DOC:VOLUME.2X
*MCS650XASR        MOS Tech 6500 Assembler CCMemo 422
*METATRAN          DI-3000 Metafile Xfr.  MTS Volume 2; CCMemo 463
*MNETTOPDS         Plot System            *PDSTOMNET, MTS Volume 2
*MPS               Numerical              MTS Volume 2
*MTSBIBLIOGRAPHY   Documentation Info.    MTS Volume 2
*MVC               Record Rearrangement   MTS Volume 2
*M6800ASR          Motorola 6800 Assembler CCMemo 422
*M6809ASR          Motorola 6809 Assembler CCMemo 422
*NETCOPYMAC        MTS Command Macros     MTS Volume 2
*NETMAILHELP       Remote Message Info.   MTS Volume 2
*NETMAILSCHEDULE   Remote Message Info.   MTS Volume 2
```

```
  *NETSERVERS.W     Server information      MTS Volume 1
  *NETWORKRATES     Rate Information        MTS Volume 2
  *NIM              Demonstration           DOC:VOLUME.2X
  *OBJLIST          Object Module Utility   MTS Volume 2
  *OBJSCAN          Object Module Utility   MTS Volume 2
  *OBJUTIL          Object Module Utility   MTS Volume 2; MTS Volume 5
  *OLDFIO           FORTRAN                 None
  *OSMAC            OS Macro Library        MTS Volume 2
  *OVERDRIVE        FORTRAN                 MTS Volume 2; MTS Volume 6
  *PAGEFONTCONVERT  Page Printer Output     MTS Volume 2
| *PAGEPR           Page Printer Output     MTS Volume 2; R1038
  *PASCAL/JB        Same as *PASCALJB
  *PASCAL/VS        Same as *PASCALVS
  *PASCALJB         Pascal/JB               MTS Volume 2; MTS Volume 20
  *PASCALJBINCLUDE  Pascal/JB               *PASCALJB, MTS Volume 2
  *PASCALJBLIB      Pascal/JB               *PASCALJB, MTS Volume 2
  *PASCALJBSYSLIB   Pascal/JB               *PASCALJB, MTS Volume 2
  *PASCALTIDY       Pascal                  MTS Volume 2; MTS Volume 20
  *PASCALVS         Pascal/VS               MTS Volume 2; MTS Volume 20
  *PASCALVSINCLUDE  Pascal/VS               *PASCALVS, MTS Volume 2
  *PASCALVSLIB      Pascal/VS               *PASCALVS, MTS Volume 2
  *PASCALVSSYSLIB   Pascal/VS               *PASCALVS, MTS Volume 2
  *PDP11ASR         PDP 11 Assembler        CCMemo 422
  *PDSTOMNET        Plot System             MTS Volume 2
  *PEXIT            360/370 Assembler       MTS Volume 2; MTS Volume 14
  *PFORT            FORTRAN Verifier        MTS Volume 2; CCMemo 406
  *PGF              Page Printer Fonts      *PAGEPR, MTS Volume 2
  *PIL              PIL/2                   MTS Volume 12
  *PL/C             Same as *PLC
  *PL/1             Same as *PL1
  *PL/1LIB          Same as *PL1LIB
  *PL/1SCAN         Same as *PL1SCAN
  *PL/1TIDY         Same as *PL1TIDY
  *PLC              PL/I Cornell Compiler   MTS Volume 2; MTS Volume 7
  *PLOT             Printer Plotting        DOC:VOLUME.2X
  *PLOTSEE          Plot System             MTS Volume 2
  *PLOTSTACK        Plot System             MTS Volume 2
  *PLOTSYS          Plot System             MTS Volume 2; MTS Volume 11
  *PLOTTIMES        Plot Information        MTS Volume 2
  *PLUS             PLUS Programming Lang.   *PLUS, MTS Volume 2
  *PLUS.OBJLIB      PLUS Programming Lang.   *PLUS, MTS Volume 2
  *PLUS.SOURCELIB   PLUS Programming Lang.   None
  *PLUS11           PLUS Programming Lang.   *PLUS, Volume 2
  *PL1              Same as *PL1F
  *PL1F             PL/I (F)                MTS Volume 2; MTS Volume 7
  *PL1FLIB          PL/I (F)                *PL/I, MTS Volume 2; MTS
                                            Volume 7
  *PL1LIB           Same as *PL1FLIB
  *PL1OPT           PL/I Optimizing Comp.   MTS Volume 2; MTS Volume 7
  *PL1OPTLIB        PL/I Optimizing Comp.   *PL1OPT, MTS Volume 2; MTS
                                            Volume 7
  *PL1SCAN          PL/I (F)                MTS Volume 2; MTS Volume 7
  *PL1TIDY          PL/I (F)                MTS Volume 2; MTS Volume 7
```

```
  *PL360            PL360                 CCMemo 75
  *PL360LIB         PL360                 CCMemo 75
  *PRESCAN          TEXT360               MTS Volume 15
  *PRINT            TEXT360               MTS Volume 15
  *PRODUCTSUPPORT   Support information   MTS Volume 2
  *PROFORT          FORTRAN Exec. Profiler MTS Volume 6
  *PROLOGC          Prolog language       MTS Volume 2; CCMemo 478
  *PROLOGW          Prolog language       MTS Volume 2; CCMemo 477
  *RATES            Rate Information      MTS Volume 2
  *RATFOR           FORTRAN Preprocessor  MTS Volume 2
  *REBATEFORM       Rebate Request Form   None
  *REDUCE           Symbolic Mathematics  MTS Volume 2
| *REQUESTACCOUNTS  Request Account Info. None
  *RESTORE          File Restoration      MTS Volume 2
  *RG               Report Generator      MTS Volume 2; CCMemo 437
| *RMLOG            Resource Manager Log  MTS Volume 2; R1037
  *SCRAMBLE         File Security         DOC:VOLUME.2X
  *SDL              Survey Definition Lang. MTS Volume 2, CCMemo 452
  *SENSEDSECT       Subroutine Dsect      MTS Volume 4
  *SIDEBYSIDE       Page Printer Output   MTS Volume 2
  *SIDEDATE         File Comparison       DOC:VOLUME.2X
  *SIGSETUP         Sigfile processing    MTS Volume 2
  *SIMSCRIPT2       SIMSCRIPT II          DOC:VOLUME.2X
  *SIM2ASM          SIMSCRIPT II          DOC:VOLUME.2X
  *SIM2CERRORS      SIMSCRIPT II          None
  *SIM2COMP         SIMSCRIPT II          DOC:VOLUME.2X
  *SIM2ERRORS       SIMSCRIPT II          DOC:VOLUME.2X
  *SIM2LIB          SIMSCRIPT II          DOC:VOLUME.2X
| *SITES            Campus Comp. Sites Info. MTS Volume 2
  *SKELETON         XPL                   DOC:VOLUME.2X
| *SLIDEQ           Graphics              MTS Volume 2; R1048
  *SLIP             SLIP                  MTS Volume 8
  *SLIPDEC          SLIP                  MTS Volume 8
  *SNAP             Text Processing       DOC:VOLUME.2X
  *SNOBOL4          SNOBOL4               MTS Volume 2; MTS Volume 9
  *SNOBOL4B         SNOBOL4               MTS Volume 2; MTS Volume 9
  *SNOSTORM         SNOBOL4               MTS Volume 2; MTS Volume 9
  *SOFTWARECHARGES  Rate Information      MTS Volume 2
  *SORT             Sort/Merge Program    MTS Volume 2; MTS Volume 5
  *SPIRES           Database Management   MTS Volume 2
  *SPITBOL          SNOBOL4               MTS Volume 2; MTS Volume 9
  *SPITDEBUG        SNOBOL4               MTS Volume 2; MTS Volume 9
  *SPITERR          SNOBOL4               *SPITBOL, MTS Volume 2; MTS
                                          Volume 9
  *SPITLIB          SNOBOL4               MTS Volume 2; MTS Volume 9
  *SPITSYSLIB       SNOBOL4               MTS Volume 2; MTS Volume 9
  *SS.FTN.EX        Screen Routines Example MTS Volume 4
  *SS.PAS.EX        Screen Routines Example MTS Volume 4
  *STATUS           Signon ID Information DOC:VOLUME.2X
  *ST360            System/360 Simulator  DOC:VOLUME.2X
  *SURVEY           User Survey           None
  *SURVEYCOMMENTS   User Survey Comments  None
  *SURVEYRESULTS    User Survey Results   None
```

```
    *SYMBOLS          Symbol Information      MTS Volume 2
    *SYSMAC           System Macro Library    MTS Volume 2; MTS Volume 14
    *TABEDIT          Tab-Editing             DOC:VOLUME.2X
    *TALLY            Instruction Counting    DOC:VOLUME.2X
    *TANGO            Tango Programming Lang.  MTS Volume 2
    *TANGOLIB         Tango Library           *TANGO, MTS Volume 2
    *TAPECOPY         Magnetic Tapes          MTS Volume 2; MTS Volume 19
    *TAPEDUMP         Magnetic Tapes          MTS Volume 2; MTS Volume 19
    *TAPEFIXER        Magnetic Tapes          MTS Volume 2; MTS Volume 19
  | *TAPERETRIEVE     Magnetic Tapes          MTS Volume 2; T7012
  | *TAPES            Magnetic Tapes          MTS Volume 2; T7012
  | *TAPESUBMIT       Magnetic Tapes          MTS Volume 2; T7012
    *TAXIR            Database Management     MTS Volume 2; Taxir Primer
    *TDAY88FSMUTIL    T-Day Translation       None
    *TDAY88TRANS      T-Day Translation       MTS Volume 2; CCMemo 480
    *TELLABANK        Tell-A-Graf Graphics    MTS Volume 2, CCMemo 450
    *TELLAGRAF        Tell-A-Graf Graphics    MTS Volume 2, CCMemo 450
    *TELLALIB         Tell-A-Graf Graphics    CCMemo 450
    *TERMINALS        Terminal Information    None
    *TERMLIST         Terminal Information    MTS Volume 2
    *TERMTYPES        Terminal Information    MTS Volume 2
    *TEX              Text Processing         MTS Volume 2, CCMemo 810
    *TEXLEVELCHECK    Text Processing (TEX)   CCMemo 810
    *TEXTFORM         Text Processing         MTS Volume 2
    *TEXT360          Text Processing         MTS Volume 15
    *TIDY             FORTRAN                 DOC:VOLUME.2X
    *TIME             Time of Day             DOC:VOLUME.2X
    *TRANSNOBOL       SNOBOL3-SNOBOL4 Conv.   MTS Volume 9
    *TYPEIT           Text Processing         None
    *TYPEQ            Phototypesetter         MTS Volume 2
    *T360MASTCONV     TEXT360                 MTS Volume 15
    *T90T360          TEXT360                 None
    *UAM              System Information      None
    *UMIST            UMIST                   DOC:VOLUME.2X
    *UNEDIT           File Comparison         MTS Volume 2
    *UNIXTAR          Unix Tape Processing    None
    *UNLINKER         Data Set Utility        CCMemo 449
    *UPDATE           File Updating           DOC:VOLUME.2X
    *USERDIRECTORY    Message System          MTS Volume 2; MTS Volume 23
    *UTILISP          LISP                    MTS Volume 2; MTS Volume 22
    *VALIDATEFILE     File Checking           MTS Volume 2
    *VSS              VSS Environment         MTS Volume 2, CCMemo 448
    *VSSEMPTYPDS      VSS Data Set Utility    CCMemo 449
    *VSSPDSBUILD      VSS Data Set Utility    CCMemo 449
    *VSSPDSCOMPRESS   VSS Data Set Utility    CCMemo 449
    *VSSPDSCONVERT    VSS Data Set Utility    CCMemo 449
    *VSSPDSLIST       VSS Data Set Utility    CCMemo 449
    *WATBOL           Waterloo COBOL          MTS Volume 2
    *WATBOLLIB        Waterloo COBOL          *WATBOL, MTS Volume 2
    *WATBOLLIBGEN     Waterloo COBOL          MTS Volume 2
    *WATFIV           Waterloo FORTRAN        MTS Volume 2; MTS Volume 6
    *WATGENLIB        Waterloo FORTRAN        MTS Volume 6
    *WATLIB           Waterloo FORTRAN        *WATLIB, MTS Volume 2; MTS
```

```
                                        Volume 6
*WBASIC         Waterloo BASIC          MTS Volume 2
*WIREWRAP       Hardware Logic Design   MTS Volume 2; CCMemo 267
*WORKSTATIONS   Terminal Information    MTS Volume 2
*WWICMAC        Hardware Logic Design   CCMemo 267
*WWLABMAC       Hardware Logic Design   CCMemo 267
*WWLIB          Hardware Logic Design   CCMemo 267
*WWSQGRD        Hardware Logic Design   CCMemo 267
*WWSTGRD1       Hardware Logic Design   CCMemo 267
*WWSTGRD2       Hardware Logic Design   CCMemo 267
*XALGOLW        ALGOL W                 MTS Volume 16
*XCOM           XPL                     DOC:VOLUME.2X, CCMemo 230
*XMON           XPL                     CCMemo 230
*XPL            XPL                     DOC:VOLUME.2X; CCMemo 230
*XPLCOMPILER    XPL                     *XPL, MTS Volume 2; CCMemo 230
*XPLGO          XPL                     DOC:VOLUME.2X; CCMemo 230
*XPLIBRARY      XPL                     DOC:VOLUME.2X; CCMemo 230
*Z80ASR         Zilog X-80 Assembler    CCMemos 422 and 423
*1              L6 Macro Library        None
*1ASR           PDP-1 Assembler         CCMemo 329
*11ASR          PDP-11 Assembler        MTS Volume 2; CCMemo 286
*1130ASM        IBM 1130 Assembler      DOC:VOLUME.2X
*8ASR           PDP-8 Assembler         CCMemo 329
*8LINK          PDP-8 Linkage Editor    CCMemo 329
*9ASR           PDP-9 Assembler         CCMemo 329
```

SUBJECT CATEGORIES OF PUBLIC FILES

In an effort to aid people in finding programs that may be useful in their work, a number of subject categories have been defined. Each category consists of a type of activity a user might be doing, and under each category are listed the names of the public files that might be of use in that activity. The file names are listed alphabetically.

Assembly Language Usage

        *ASMH
        *ASMTIDY
        *ASSIST
        *MACUTIL
        *PEXIT
        *SYSMAC

C Language

        *CBELL
        *CBELLINCLUDE
        *CBELLLIB
        *C87
        *C87INCLUDE
        *C87LIB

Database

        *RG
        *SPIRES
        *TAXIR

File Manipulation

        *APC
        *CDUPDATE
        *COMBINE
        *COMPARE
        *ENCRYPT
        *FILESCAN
        *FS
        *RESTORE
        *UNEDIT
        *VALIDATEFILE

     FORTRAN Usage

          *DAVE
          *FORTRANVS
          *FTN
          *FTNTIDY
          *IF66
          *IF77
          *PFORT
          *PROFORT
          *RATFOR
          *WATFIV
          *WATGENLIB
          *WATLIB

     Graphics

          *CCQUEUE
          *DI3000.xxx
          *HPGLPLOT
          *IG
          *LOCALPLOT
          *METATRAN
          *MNETTOPDS
          *PDSTOMNET
          *PLOTSEE
          *PLOTSTACK
          *PLOTSYS
          *SLIDEQ
          *TELLABANK
          *TELLAGRAF

     Information

          *CAMPUSZIPS
          *CCBIBLIOGRAPHY
          *CCDELIVERY
          *CCPHONES
          *CCPUBLICATIONS
          *CCRESHOURS
          *CCSOFTWARE
          *COMPUCALENDAR
          *CONSULTINGHOURS
          *CONSULTINGSCHED
          *CRCBOOKLIST
          *CRCSOFTWARE
| *CRISPINFO
          *DOCINFO
          *FORUM
          *MTSBIBLIOGRAPHY
          *NETMAILSCHEDULE

```
        *NETWORKRATES
        *PRODUCTSUPPORT
        *RATES
|       *REQUESTACCOUNTS
|       *SITES
        *SOFTWARECHARGES
        *TERMLIST
        *TERMTYPES
```

Language Processors

```
        *ALGOLW
        *APL
        *ASMH
        *ASSIST
        *BASICUM
        *CBELL
        *COBOLVS
        *CSMP
        *CSMPTRAN, *CSMPEXEC
        *C87
        *EXPL
        *FORTRANVS
        *FTN
        *GASP
|       *GPSSH2
        *ICON
        *IF66
        *IF77
        *PASCALJB
        *PASCALVS
        *PLC
        *PLUS
        *PLUS11
        *PL1F
        *PL1OPT
        *PROLOGC
        *PROLOGW
        *SNOBOL4
        *SNOBOL4B
        *SPITBOL
        *TANGO
        *UTILISP
        *WATBOL
        *WATFIV
        *WBASIC
        *11ASR
```

   Macro Libraries

        *CMDMACLIB
        *MACUTIL
        *NETCOPYMAC
        *OSMAC
        *SYSMAC

   Microcomputers

        *KERMIT
        *MACBINHEX
        *MACPAINT4045
        *MACWRITE9700

   Numerical and Mathematical

        *MPS
        *REDUCE

   Object Module Processing

        *ENDJUNK
        *LCS
        *LINKEDIT
        *OBJLIST
        *OBJSCAN
        *OBJUTIL

   Pascal Usage

        *PASCALJB
        *PASCALJBINCLUDE
        *PASCALJBLIB
        *PASCALJBSYSLIB
        *PASCALTIDY
        *PASCALVS
        *PASCALVSINCLUDE
        *PASCALVSLIB
        *PASCALVSSYSLIB

   PL1 Usage

        *PLC
        *PL1F
        *PL1FLIB
        *PL1OPT
        *PL1OPTLIB

```
        *PL1SCAN
        *PL1TIDY


    SNOBOL4 Usage

        *SNOBOL4
        *SNOBOL4B
        *SNOSTORM
        *SPITBOL
        *SPITDEBUG
        *SPITERR
        *SPITLIB
        *SPITSYSLIB


    Subroutine Libraries

        *CBELLLIB (see *CBELL)
        *CSMPLIB
        *C87LIB
        *EXPLIB (see *EXPL)
        *ICONLIB (see *ICON)
        *LIBRARY
        *PASCALJBINCLUDE (see *PASCALJB)
        *PASCALJBLIB (see *PASCALJB)
        *PASCALJBSYSLIB (see *PASCALJB)
        *PASCALVSINCLUDE (see *PASCALVS)
        *PASCALVSLIB (see *PASCALVS)
        *PASCALVSSYSLIB (see *PASCALVS)
        *PL1FLIB (see *PL1F)
        *PL1OPTLIB (see *PL1OPT)
        *SPITLIB
        *SPITSYSLIB
        *SYMBOLS
        *TANGOLIB (see *TANGO)
        *WATBOLLIBGEN
        *WATLIB (see *WATFIV)
        *WATGENLIB


    Tape Usage

        *CDUPDATE
        *CMSTAPELOAD
        *CMSTAPESCAN
        *DEDIT
        *FS
        *LABEL
        *LABELSNIFF
        *TAPECOPY
        *TAPEDUMP
        *TAPEFIXER
```

       *TAPERETRIEVE
       *TAPES
       *TAPESUBMIT
       *UNIXTAR

Text Processing

       *BIBTEX
       *DITTO
       *DVIAPS
       *DVIXER
       *LATEX
       *MACPAINT4045
       *MACWRITE9700
       *TEX
       *TEXTFORM

Utility Programs

       *ANALYSIS
       *APLFONTCONVERT
       *ARC
       *AUTOSTART
       *BCDEBCD
       *CKID
       *CKSIG
       *CLPARSEGEN
       *CMS
       *COLLATE
       *EBCDBCD
       *FLOWCHART
       *LABELS
       *LOCALPRINT
       *LOGPURGE
       *MVC
       *PAGEFONTCONVERT
       *PAGEPR
| *RMLOG
       *SDL
       *SIDEBYSIDE
       *SIGSETUP
       *SORT
       *SYMBOLS
       *TDAYTRANS
       *USERDIRECTORY
       *VSS
       *WIREWRAP

January 1987

OTHER AVAILABLE FILES

   This volume describes only "public files"--files maintained by the
Computing Center for general use. Public file names are distinguished
by an asterisk "*" as the first character of their name. In addition to
public files, many groups on campus maintain collections of files for
general use.   These are read-only shared files, and are accessed by
prefixing the file name with the Computing Center ID that owns the file
and a colon ":". For example, STAT:CATALOG. The Computing Center ID
used for such communication purposes is usually some mnemonic name, such
as STAT. Some of these are:

   BSAD - Graduate School of Business Administration

   CAP - Computer Assistance Program, Department of Political Science

   ECON - Department of Economics (TROLL, SHAZAM, TIMESERIES)

   EDUC - School of Education

   ILIR - Institute of Labor and Industrial Relations (MICRO, MIST)

   ISR - Institute for Social Research (OSIRIS-IV)

   NAAS - Numerical Analysis and Applications Software

   NEW - New versions of Computing Center software

   PCn - Computing Center Micro Consultants Group (Personal computer
          software - currently, the IDs PC, and PC1 through PC6 are
          being used)

   OLD - Old versions of Computing Center software

   SPSS - Department of Sociology

   STAT - Statistical Research Laboratory (MIDAS, TEXTEDIT)

   UNSP - Unsupported software
          See below.

Specific reference will be made in other volumes of the manual. Several
departments also maintain such communication CCIDs. These are usually
for courses within the department, but often may contain files of
general interest.

The Computing Center ID UNSP is part of an effort to gather a number
of unsupported programs and subroutines into one location.   This
unsupported software is being made available under UNSP rather than in
public files because the Computing Center does not have the resources
(people, time, or money) to completely insure its quality or to provide
continuing maintenance.   Many of these programs and subroutines repre-
sent interim solutions to particular problems which will be replaced
with supported software as better solutions are developed.

As the name UNSP suggests, this software is not actively supported by
the Computing Center staff. This means that there are no guarantees as
to its reliability, performance, or continued availability, no counsel-
ing is available beyond that normally provided for user programs, and no
rebates will be given for errors caused by the operation of unsupported
software. (It should be noted, however, that before any software is
made available under UNSP, a member of the Computing Center staff will
have done minimal testing and determined that the program does what it
claims to do for the common cases.) The file UNSP:CATALOG may be copied
to obtain a list of the programs and subroutines currently available
together with a short description and directions for obtaining addition-
al documentation.

January 1987

PUBLIC FILE DESCRIPTIONS

   Following are the descriptions of the public files available in MTS.
These descriptions are designed to be used as reference for using the
files; full explanatory descriptions are not provided here.   Introduc-
tions and detailed usage guides will be found in the other volumes, and
in general the public file descriptions are repeated there.

January 1987

<u>*ALGOLW</u>

Contents: The MTS version of the ALGOL W compiler.

Purpose: To compile ALGOL W programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:
                 SCARDS - program or external procedure to be compiled
                          (exactly one ALGOL W program or external proce-
                          dure is allowed).
                 SPRINT - output of compilation listing.
                 SERCOM - diagnostic messages produced by the compiler.
                 SPUNCH - object module of the compiled program.

Return Codes: 0 - Successful return.
               4 - Error return.
               8 - Error return.
               16 - Execution suppressed.

Description: This compiler is the MTS version of the ALGOL W compiler
                 written at Stanford University. The definition of the
                 language and the details for its use in MTS are given in
                 MTS Volume 16, <u>ALGOL W in MTS</u>.

Examples: $RUN *ALGOLW SCARDS=ALGOLWSOU SPUNCH=ALGOLWOBJ PAR=DECK

                     In the above example, the ALGOL W source program is
                     read from the file ALGOLWSOU and the object module
                     is written to the file ALGOLWOBJ.

                 $RUN *ALGOLW
                 /COMPILE
                    source program
                 /EXECUTE
                    data
                 /STOP

                     The above example illustrates the method of running
                     ALGOLW in compile-and-execute mode. By default, a
                     source listing is written to the file -AWLIST.

January 1987

*ANALYSIS
——————

Contents: A program for analyzing the statistics produced on
                magnetic tape by the data collection facility of MTS (the
                STAT job).

Purpose: To aid in the processing of the detailed statistical data
                records on magnetic tape by the UMMPS data collection
                facility.

Use: The program is invoked by a $RUN command specifying
                *ANALYSIS (or in some cases, *ANALYSIS(0)) as the object
                file.

Program Key: *ANALYSIS

Logical I/O Units Referenced:
                SCARDS - the statistical tape to be processed.
                SPRINT - all output produced by the program, in machine
                         carriage-control format.

Return Codes: None.

Description: The data collection facility of MTS (the STAT job)
                produces a very detailed chronological history of certain
                events of interest occurring in the system. Paging
                events, input/output operations, assignments of proces-
                sors to tasks, etc., may all be recorded. The data
                collection procedure must be invoked from the operator's
                console. It cannot be invoked by a user at a terminal or
                by a batch job.

                Once a statistics tape has been produced, *ANALYSIS may
                be used to aid in any data reduction necessary. It has a
                large number of optional features controlled by the PAR
                field of the $RUN command. These options allow direct
                printout of the statistics on the tape, at varying levels
                of detail, or selection of statistics relating only to
                certain tasks or certain aspects of system activity. In
                addition, user-written subroutines may be loaded with
                *ANALYSIS. *ANALYSIS will pass data to these subroutines
                for further processing.

                The use of the data collection facility and *ANALYSIS is
                a fairly complex procedure requiring more documentation
                than it is possible to reproduce here. For a complete
                description of the use of *ANALYSIS (and the collection
                of data) the following references are given.

January 1987

(1) Computing Center Memo 393, "Data Collection Facility."

This contains a complete operating guide to both the data collection facility and *ANALYSIS. The level of detail is sufficient to allow the use of the facilities by a user with a familiarity of the structure of the UMMPS/MTS system.

(2) Program Behavior and Control in Virtual Storage Computer Systems, CONCOMP Project Technical Report 4, by Tad B. Pinkerton.

The appendix of this report contains a good discussion of the philosophy of the data collection facility and the overhead incurred in its use.

(3) Performance Analysis of Multilevel Paging Hierarchies, James A. Hamilton, Doctoral Dissertation, University of Michigan.

An appendix to this dissertation updates the overhead analysis in (2).

*APC

Contents: The "all-purpose compare" program.

Purpose: To test for equality between records of two files.

Use: The program is invoked by the $RUN command.

Program Key: *APC

Logical I/O Units Referenced:
                SPRINT - printed output.
                SERCOM - diagnostic messages.
                GUSER - user responses in conversational mode.
                0   - file or device to be compared.
                1   - file or device to be compared.

Return Codes: 0 - Both files or devices are equal.
                4 - One or more records are unequal, or they may be all
                     equal but their lengths or line numbers are not and
                     LENCHK or LNRCHK option is ON.
                8 - An input or fatal error has occurred.

Description: This program is primarily used to test for equality
                between records of two separate files   or   devices for
                verification.   Other programs that indicate differences
                between files or devices are *AMENDS (to generate MTS
                command mode lines), *DOWNDATE (to generate *UPDATE
                control cards), *SIDEDATE (to compare two *UPDATE command
                streams), and *UNEDIT (to produce context editor control
                cards).

                The files or devices to be compared are read from the
                logical I/O units 0 and 1.   A one-to-one positional
                comparison of all characters is made on each record from
                the two files or devices. If two records being compared
                are of unequal lengths and LENCHK is not specified, the
                comparison is made as if the shorter record is padded
                with blanks.   The FILES parameter may be specified so
                that a certain number of or all files of the two tapes
                attached to the logical I/O units 0 and 1 can be
                compared.

                The all-purpose compare program generally compares the
                two files or devices record by record. However, instead
                of waiting for the user action after a mismatch, the
                program can run in the synchronization mode when the user
                enables the SYNCH option.   Then after a mismatch is
                discovered, the program will attempt to look ahead and to
                resynchronize, i.e., to match a specified number of

consecutive records of the two files. This number
depends on the parameters BREAK, MAXCOMP, MAXDEL, and
SLOPE. MAXCOMP is the maximum number of records that the
program will compare in attempting to resynchronize;
MAXDEL is the maximum number of consecutive records that
will be deleted from one of the two files in attempting
to resynchronize; BREAK and SLOPE control the actual
number of records that must match. The synchronization
mode is especially useful in batch where there is no
interaction on the part of the user.

Parameters: The following parameters may be specified in the PAR
field of the $RUN command. The parameters must be
separated by commas or blanks. Some of the parameters
(as indicated in the list below) may be negated by
prefixing with "NO", "N", "-", or "˜". Minimum accept-
able abbreviations are underlined.

    BREAK=n   "n" and "m" (from the SLOPE parame-
ter) control the number of records
that must match in the two files
before a "resynch" is recognized. If
"d" is the size in deleted records
preceding the resynch, then the match
must be "n+(d-n)/m" records long,
unless "d<n", in which case, "d" is
used. If the result is greater than
the value of MAXCOMP, then MAXCOMP is
used. The defaults are n=3 and m=7.

    EBCD   Print two erroneous records in EBCDIC
format. This is the default.

    FILES={n|ALL}   If specified, "n" or all files on two
tapes, designated by the units 0 and
1, are compared. The default is ALL.

    HEX   Print two erroneous records in hexa-
decimal format. The parameter EBCD
forces the program to print in EBCDIC
instead of hexadecimal format.

    LIMIT=n   This is enforced only in batch mode.
If the number of errors exceeds "n",
the program will be terminated. The
default is 10.

    [NO]LENCHK   If specified, line lengths are com-
pared for equality. The default is
NOLENCHK.

[NO]LNRCHK	If specified, line numbers of records are compared for equality. The default is NOLNRCHK.
MAXCOMP=n	"n" is the maximum number of records to be compared between the two files before deciding that a resynch has been found. The actual number of records compared depends on the size of the preceding deletion and the value of BREAK and SLOPE parameters. The default is 25.
MAXDEL=n	"n" is an estimate of the maximum number of records to be deleted in any one deletion from one file to the other file. Specifying a value slightly larger than the number of records actually deleted will greatly reduce the cost of running the program. If "n" is too small, the comparison output to SPRINT will be valid, but much longer than necessary. The default is 500.
PADCHAR=xx	When two records have unequal lengths, the shorter record is padded on the right with a character of the value of two hexadecimal digits "xx". The default is a blank, hexadecimal 40.
SLOPE=m	See the BREAK parameter.
[NO]SYNCH	If the synchronization mode is on, the program will attempt to compare a certain number of records according to BREAK, MAXDEL, MAXCOMP, and SLOPE parameters. This will result without the user's interaction. The default is NOSYNCH.
[NO]TRIM	Input records are trimmed off at right. The default is NOTRIM.
X	This is same as the HEX parameter.

All-Purpose Compare Command Language

If neither logical I/O unit 0 nor 1 is specified, APC goes into command
mode. Otherwise, it starts out comparing the two files. When it finds
a mismatch, it reports this and lapses into command mode. Note: In
batch, commands are not implemented; the user must specify the units
and/or parameters.

The following is the list of available APC commands. The uppercase
portion of command names and keywords indicates minimum abbreviations
that may be used.   A command line beginning with an asterisk "*" is
considered to be a comment and is not processed.   A command line
beginning with a dollar sign "$" is assumed to be an MTS command; it
will be executed and APC will regain control. All APC commands start
with the first nonblank character, which need not start at position 1.
At least one blank should separate the command name and the first
operand, and there must be a blank or a comma between any two operands.

Command: BACKward [{0|1|BOTH}] [n]

Example: BACKWARD 0 15

Explanation: This command skips backwards "n" records on either
                logical unit 0 or 1 or both. If "n" is negative,
                forward-spacing takes place. The default for "n" is 1.
                A CONTINUE command will start comparing.

Command: COLUMN [{0|1|BOTH}] [(m,n)]

Example: COLUMN (1,72)

Explanation: This command sets the left and right columns for either
                unit 0 or 1 or both between which APC may compare. The
                default is m=1 and n=32767 for both units.

Command: COMMENT comments

Example: COMMENT - Now we print two unequal records

Explanation: The COMMENT command is useful for documenting sequences
                of commands given to APC. Note that command lines
                beginning with an asterisk are also treated as comments.

Command: COMPARE FDname0 FDname1 [HOLD] [Visual]

Example: COMPARE MAC MACLIB

Explanation: The COMPARE command attaches two files or devices to the
                unit 0 and 1. If HOLD is specified, the comparison is

38 *APC

not started at once; instead, APC gets back into command
mode. A CONTINUE command starts comparing these two. If
VISUAL is specified, the units are put directly in visual
mode.

Command: CONTINUE [{COL|COL0|COL1}={m|*} | {SKIP|SKIP0|SKIP1}=n]

Example: CONTINUE

Explanation: This continues the comparison with the next character (or
                the character at the column "m" if COL is specified or at
                the current column pointer plus "n" if SKIP is speci-
                fied).   COL=* compares the current characters. Alterna-
                tively, COL0 (for unit 0), COL1 (for unit 1), SKIP0 (for
                unit 0), and SKIP1 (for unit 1) may be set for unit 0 or
                1 instead of both units.   If COL=m or SKIP=n is not
                specified, the default is the next column.

Command: DISPLAY [item] ...

Example: DISPLAY PFKEYS

Explanation: This command displays various things. Things that can be
                displayed currently are PFKEYS and STATUS.

                DISPLAY PFKEYS

                    The action of DISPLAY PFKEYS depends on whether the
                    command is issued while in visual mode or while in
                    line mode. If issued in line mode, the current
                    PFkey definitions are displayed. If issued in
                    visual mode, the user is placed in the editor in
                    visual mode, and the PFkey definitions can be
                    viewed, edited, etc.

                DISPLAY STATUS

                    DISPLAY STATUS is a synonym for the STATUS command.

Command: EDit {0|1} [cmnd]

Example: EDIT 1 visual

Explanation: This command is the same as issuing the MTS command $EDIT
                for the given unit.

Command: EXPLAIN [item] ...

Example: EXPLAIN STATUS

Explanation: This command gives explanations of the items specified in
                full-screen help. The following items can be explained:
                $, *, ?, BACK, BREAK, COLUMN, commands, COMMENT, COMPARE,
                CONTINUE, description, EBCD, everything, example, EX-
                PLAIN, FILES, FWD, HELP, HEX, LENCHK, LIMIT, LINE,
                LNRCHK, MAXCOMP, MAXDEL, MCMD, MTS, parameters, PADCHAR,
                PRCOL, PRINT, PRLEN, READ, RETURN, SCAN, SET, SLOPE,
                STATUS, STOP, SYNCH, TRIM, VISUAL.

Command: {FORward|Fwd} {0|1|BOTH} [n]

Example: FORWARD BOTH 5

Explanation: This command skips "n" records on either logical unit 0
                or 1 or both. If "n" is negative, backspacing takes
                place. The default for "n" is 1. A CONTINUE command
                will start comparing.

Command: HELP

Example: HELP

Explanation: This command is a synonym for the EXPLAIN command.

Command: LINE {0|1|BOTH} lnr

Example: LINE 0 1.55

Explanation: A record is read from the unit 0 or 1 at the line number
                "lnr".   In addition to its actual line number value,
                "lnr" can have the value of *F, *L, *N, and *P for the
                first line, last line, next line, and previous line,
                respectively, just as in the editor.

Command: MCMD cmnd

Example: MCMD $LIST FYLE(20,25)

Explanation: "cmnd" must be a valid MTS command. It will be executed,
                and control is returned to APC. Alternatively, the user
                may issue the MTS command directly in the command mode by
                prefixing it with a dollar sign, e.g., $EMPTY QQ.

Command: MTS [cmnd]

Example: MTS

Explanation: "cmnd", if specified, must be a valid MTS command. It
                will be executed, and APC may be reentered by a $RESTART
                MTS command.

Command: PRINT [{0|1|BOTH}] [{HEX|X|EBCD}] [LEN=l] [COL={*|c}]
                      [{COUNT|CNT}=n]

Example: PRINT EBCD LEN=56 C=1

Explanation: This prints unit 0 or 1 or both on I/O unit SPRINT in
                either hexadecimal (see the previous descriptions of the
                HEX and X parameters) or EBCDIC (see the previous
                description of the EBCD parameter) format with length =
                "l", starting at either the current column "*" or the
                specified column "c". If "0" or "1" is not specified,
                current records from both units are printed.   The
                defaults are EBCD, LEN=32767, and COL=1. These defaults
                may be overridden by the SET command. If COUNT or CNT is
                set to "n", then "n" records of unit 0 or 1 are printed.
                This is similar to listing records.

Command: READ [{0|1|BOTH}]

Example: READ 0

Explanation: If "0" and "1" are not specified, both units will be
                read.   Otherwise, a record from the specified unit is
                read. Then, APC starts comparing.

Command: RETURN

Example: RETURN

Explanation: This terminates the processing and control returns to the
                caller (normally MTS command mode).

Command: SCAN {0|1} [{BACK|FWD}]

Example: SCAN 0

Explanation: This command scans the logical unit 0 or 1 for the
                matching record from the other unit. If successful, it
                prints the line number. If not, a message is printed.
                If BACK parameter is specified, the scanning will be
                backwards instead of forwards.

Command: SET option ...

Example: SET HEX=ON, PRLEN=56, PRCOL=*

Explanation: The SET command is used to alter the status of various
                APC options, default switches or default characters.
                Options must be separated by blanks or commas; there must
                be no blanks within any option.

        BREAK=n   Default: 3

            "n" and "m" (from the SLOPE option) control the
            number of records that must match between the two
            files before a "resynch" is recognized. If "d" is
            the size in deleted lines preceding the resynch,
            then the match must be "n+(d-n)/m" records long,
            unless "d<n", in which case, "d" is used. If the
            result is greater than the value of MAXCOMP, then
            MAXCOMP is used. The defaults are n=3 and m=7.

        FILES={n|ALL}   Default: ALL

            The FILES option is used to compare "n" or ALL files
            of two tapes, 0 and 1. Two consecutive end-of-files
            terminate the processing.

        HEX={ON|OFF}   Default: OFF

            If the HEX option is ON, all records are printed in
            hexadecimal notation when the user issues the PRINT
            command and does not specify the EBCD parameter. If
            the option is OFF, records will be printed in EBCD.

        LENCHK={ON|OFF}   Default: OFF

            If the LENCHK option is ON, line lengths are
            compared. If an error occurs in the line lengths, a
            message indicating the erroneous line lengths will
            be printed. This can easily occur when MTS trims a
            record off by removing extra trailing blanks on the
            left. Line lengths are never compared if LENCHK is
            OFF or if records are unequal.

        LNRCHK={ON|OFF}   Default: OFF

            If the LNRCHK option is ON, line numbers are
            compared. If an error occurs in the line numbers
            (i.e., if they are not identical in both files), a
            message indicating the erroneous line numbers will
            be printed.   If LNRCHK is OFF or records are
            unequal, line numbers are not compared.

MAXCOMP=n   Default: 25

"n" is the maximum number of records to be compared
between two files before deciding that a resynch has
been found. The actual number of records compared
depends on the size of the preceding deletion and
the value of BREAK and SLOPE.

MAXDEL=n   Default: 500

"n" is an estimate of the maximum number of records
to be deleted in any one deletion from one file to
another file. Specifying a value slightly larger
than the number of records actually deleted will
greatly reduce the cost of running the program. If
"n" is too small, the comparison output to SPRINT
will be valid, but much longer than necessary.

PADCHAR=xx   Default: 40 (blank)

The PAD option with value in two hexadecimal digits
is used to pad shorter records before the comparison
is made. The default of the PAD option is a blank,
equivalent to hexadecimal 40.

PRCOL={*|c}   Default: 1

The PRCOL option prints the records starting at
position "c" or "*" (the current column). PRCOL=1
will print records starting at column 1.

PRLEN=l   Default: 32767

The PRLEN option specifies the length of an
erroneous record to be printed. This controls the
amount of information. The default is from the
current column to the end of record.

SLOPE=m   Default: 7

See the BREAK option.

SYNCH={ON|OFF}   Default: OFF

If SYNCH is ON, the program will attempt to synchro-
nize the records of the two files it is comparing.

TRIM={ON|OFF}   Default: OFF

The input records are usually read without trimming.
If the user desires to set the trim feature on so
that excessive blanks at right are removed, he

                         should set TRIM=ON. TRIM option should be set on
                         especially for tapes in visual mode.

Command: STATUS [{0|1|BOTH}]

Example: STATUS 0

Explanation: This command prints units 0 and/or 1 with FDnames, file
                numbers (if a tape), record numbers (if a tape), line
                numbers, record lengths, and current columns.

Command: STOP

Example: STOP

Explanation: This command terminates processing, and control is re-
                turned to the caller (normally MTS command mode).

Command: VEXEcute

Example: VEXECUTE

Explanation: This command causes APC to execute a command given on the
                current command line when in visual mode. Note that this
                command is really only effective when used as a pfkey
                command.

Command: VEXIT

Example: VEXIT

Explanation: This command causes APC to exit visual mode.

Command: VISUAL [lnr0 [lnr1]]

Example: VISUAL *F 10.5

Explanation: This command gets APC into visual mode. Line numbers may
                be specified for units 0 and 1; the defaults are the
                current line numbers.   This command should normally
                follow the COMPARE command with the word HOLD. See
                below.

The VISUAL command is supported for any device supported by the MTS
screen-support routines. Two files being compared by APC are displayed
in two columns on the screen. In addition, the cursor, control keys,
and program-function keys can be used to keep the command language to a
minimum. In the visual mode of APC, the screen has three parts:

    (1)   A "status" line (the top line of the screen) indicating the file
          names being displayed. The user can also type in the lines he
          wants to compare.
    (2)   A "line" area, divided in two columns, showing as many lines as
          possible of the two files, with corresponding line numbers.   If
          the lines are different, the line numbers are intensified. All
          nonprinting characters of lines are automatically converted to
          "?".
    (3)   A "command" line, which occupies the bottom line. Here the user
          can enter any valid APC command.

Program-function keys have the following default assignments:

    PF1:   Similar to "BACK 0 10", which moves ten previous lines of the
           first file back to the screen.
    PF2:   BACK BOTH 10. Ten lines of both files are moved back.
    PF3:   BACK 1 10. Ten previous lines of the second file are moved
           into the screen.
    PF4:   Similar to "FWD 0 10". Next ten lines of the first file are
           shown on the screen.
    PF5:   FWD BOTH 10. Next ten lines of both files are shown.
    PF6:   FWD 1 10. Next ten lines of the second file are shown.
    PF7:   FWD 0 1. Next line of the first file is shown.
    PF8:   FWD BOTH 1. Next lines of both files are shown.
    PF9:   FWD 1 1. Next line of the second file is shown.
    PF10: SCAN 0. The first file is scanned, starting from the current
           line, and terminating at a line matching the current line of
           the second file.
    PF11: CONTINUE. Both files are compared, until a mismatch is found.
    PF12: SCAN 1. The second file is scanned, starting from the current
           line, and terminating at a line matching the current line of
           the first file.

Notice that the program-function keys are arranged in three columns.
The left column is used for the first file displayed in the left column
of the screen, the right column of program-function keys is used for the
second file, and the middle column is used for both files.

The lines can be made current by moving the cursor to the desired row
displaying these lines, or by typing the line numbers in the status area
just before the file name.

The bottom lines can be used to send the APC commands (especially those
not readily available on the program-function keys), such as "COLUMN
(1,72)", "$EDIT FILE2", or "SET LNRCHK=ON".

The other control keys are:

    CLEAR = rewrite the screen,
    PA1 = interrupt (especially useful during the comparison or
        scanning),
    CNCL (or PA2) = cancel the visual mode,

        TESTREQ = set the cursor to the command area,
        ENTER = process the command, etc.

APC program-function keys are summarized below:

PF1 Back 0 10	PF2 Back both 10	PF3 Back 1 10
PF4 Fwd 0 10	PF5 Fwd both 10	PF6 Fwd 1 10
PF7 Fwd 0 1	PF8 Fwd both 1	PF9 Fwd 1 1
PF10 Scan 0	PF11 Continue	PF12 Scan 1

Example

```
#$run *apc
#EXECUTION BEGINS
@compare *sysmac *asmtsysmac
 Mismatch at col 12 unit 0 *SYSMAC lnr 2.5 and unit 1 *ASMTSYSMAC
 lnr 2.5
@status
 Unit 0: *SYSMAC lnr 2.5 len 13 col 12
 Unit 1: *ASMTSYSMAC lnr 2.5 len 13 col 12
@print
   Unit 0: *SYSMAC lnr 2.5 len    Unit 1: *ASMTSYSMAC lnr 2.5
   13 col 12                      len 13 col 12
  IHBOPLST 268                 | IHBOPLST 265
@print hex
   Unit 0: *SYSMAC lnr 2.5 len    Unit 1: *ASMTSYSMAC lnr 2.5
   13 col 12                      len 13 col 12
  C9C8C2D6D7D3E2E340F2F6F840     C9C8C2D6D7D3E2E340F2F6F540
@column (1,8)
@continue
 Mismatch at col 7 unit 0 *SYSMAC lnr 10.5 and unit 1 *ASMTSYSMAC
 lnr 10.5
@print
   Unit 0: *SYSMAC lnr 10.5 len   Unit 1: *ASMTSYSMAC lnr 10.5
   13 col 7                       len 13 col 7
  IOPMOD   912                 | IOPMODS  909
@set synch=on
@column
@compare 0 1
   *******************    1 equal line  ***********************
       2      TWO                        2      NEW TWO
   *******************    4 equal lines ***********************
       7      SEVEN                      After line 6
```

```
   ********************    1 equal line  **********************
    After line 8                        8.25   EIGHT AND A QUARTER
                                        8.5    EIGHT AND A HALF
   ********************    2 equal lines **********************
       11      ELEVEN              After line 10
       12      TWELVE
   ********************    6 equal lines **********************
       19      NINETEEN            After line 18
   ********************    6 equal lines **********************
 Files have 7 mismatches
@stop
 9 errors
#EXECUTION TERMINATED  RC=4
```

46.2 *APC

January 1987

*APL

Contents: The IBM VS APL Version 3.0 System.

Use: The VS APL System is invoked by the $RUN command. If a
                terminal with an APL character set is being used, its APL
                character set and the VS APL procedures for input line
                editing may be invoked; otherwise, a character translit-
                eration scheme that maps the APL characters into 2- or
                3-character (ASCII or EBCDIC) combinations will be in-
                voked. VS APL will not assume that the user's terminal
                is one that will support the APL character set; the user
                must request the support via a PAR field parameter.   See
                the main APL reference, Computing Center Memo 435, "MTS
                VS APL User's Guide," for a detailed description of this
                system.

Program Key: *APL

Logical I/O Units Referenced:
                SCARDS - Source statements and commands for VS APL.
                SPRINT - Output from VS APL.

Return Codes: 0 - Successful return.
                >0 - Internal APL error.

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command. The parameter names must be
                completely specified.

                SIZE=nP   This   specifies   the   initial default clear-
                          workspace size in pages. The range of $\underline{n}$ is 5
                          to 256, inclusive.   The default is 12 pages.
                          Alternately, the size may be given in bytes by
                          specifying SIZE=n where $\underline{n}$ is the byte size
                          value. Also, WSSIZE is a synonym for SIZE.
                          Note that this parameter approximates the actu-
                          al workspace size in that the actual space
                          presented to the user will be approximately 95%
                          of the value given due to the workspace-
                          management-storage overhead associated with the
                          specification value.

                APL   This parameter instructs VS APL to invoke the
                          APL character set appropriate to the user's
                          terminal if the set is known by VS APL. Some
                          terminals have the capability of using either
                          standard (ASCII or EBCDIC) codes or APL codes
                          without user intervention. If VS APL recog-
                          nizes the user's terminal as being in this

class then it will automatically send control
codes to the terminal to cause it to use its
APL character codes. For other terminals, the
user must physically change the terminal's
operation (e.g., change a print wheel). Re-
gardless of the terminal type, if VS APL
supports it with an APL character set, then
standard APL line editing procedures may be
used. If, however, the terminal cannot be
supported in this fashion, then VS APL will use
a character transliteration scheme to allow the
user to use APL at that terminal with its
standard character set.

VS APL uses the terminal type, as established
by either the terminal answerback received at
signon time or explicitly set by the TERMINAL
device command, to determine whether the termi-
nal is capable of communicating via APL gra-
phics. The currently recognized terminal types
are:

> AJ630   AJ830   AJ860   CD1203   DTC300
>
> DTC302 GE300   GSI300   LA120   LA36
>
> LS120   TI745   T4015   X1620

See MTS Volume 4, <u>Terminals and Networks in
MTS</u>, for further information. The IBM 2741 and
Model 38 Teletype terminals may have different
APL character assignments depending on the
typing element or box mounted. Therefore, the
APL character set cannot be uniquely determined
by the terminal type. One of the following
parameters must be specified instead of the APL
parameter to resolve the terminal type.

CORR   This must be specified when an IBM 2741 with a
            CORRESPONDENCE keyboard and a 987 typing ele-
            ment is to be used; otherwise, the IBM 2741
            terminal is assumed to have a PTTC keyboard
            with a 988 typing element.

APL2,APL3 These specify the mutually exclusive APL2 and
            APL3 character assignments, respectively (e.g.,
            for a Model 38 Teletype).

<u>T</u>IME=t   This specifies a non-zero CPU time limit (in
            seconds) to be applied to this invocation of VS
            APL. Alternately, T=t may given. The value <u>t</u>
            may be either in integer or floating-point
            form. For example, TIME=3 or T=5.0E-3. When

the time limit is satisfied during the session, VS APL suspends its operation (e.g., a function being executed) and places the user in APL command mode. Thereafter, a new time limit may be set via the)TIME APL command. Also, the user can reset the time by using this command with a zero argument.

SPACE,SP   Either parameter causes the output translitera-
tion codes to be bracketed with spaces, thus improving readability at the expense of com-pactness. Also, either forces transliteration to be used.

NM   This turns off the printing of the VS APL logon
message which is normally printed prior to workspace establishment.

ECHO   At a terminal, input lines from a)SOURCE file
are not normally echoed. Specification of this parameter will force the echoing.

NOECHO,NE In batch, input lines from either a)SOURCE or
the APL input stream are echoed. Specification of this parameter will defeat the echoing. NE is an alternate for NOECHO.

NOCONTENT When VS APL loads an APL/360 workspace it does
a _format_ conversion in which all global objects (variables, functions, and groups) and their names are converted into VS APL internal for-mat. In addition, it performs a _content_ con-version in which all function statements are examined for items that require conversion in order to execute properly under VS APL. Such items are converted to their VS APL equivalent, if possible, and are noted on the conversion report. Items with no VS APL equivalent are just noted. Specification of this parameter affects _only_ the _format_ part of the conversion.

NOSUMMARY After VS APL converts an APL/360 workspace and
reports on the process, it prints a summary containing the number of functions, variables and other items. Specification of this parame-ter will defeat the printing of the summary.

Description: APL is a general-purpose computing language first defined
by K. E. Iverson in _A Programming Language_ (Wiley, 1962). IBM Corporation originally developed a conversation interpreter for this language called APL/360. This was supported in MTS at the University of Michigan until the new IBM version of APL, VS APL, was installed to

supercede APL/360 but yet provide an APL/360-to-VS APL
workspace-conversion process which would effect conver-
sions, as needed, and summarize their results. VS APL
supports new operations that are not supported by APL/
360. Moreover certain APL/360 functions or system varia-
bles do not have VS APL equivalents. For more informa-
tion, see Computing Center Memo 435, "MTS VS APL User's
Guide," and the following IBM documents:

(1) <u>APL Language</u>, GC26-3874 (Reference Text)
(2) <u>APL/360 Primer</u>, GH20-0689 (Student Text)
(3) <u>VS APL Reference Summary</u>, ST00-0602 (Reference
    card)

Examples: $RUN *APL

               $RUN *APL PAR=APL,ECHO

               $RUN *APL PAR=APL,SIZE=200P

               $RUN *APL PAR=APL2,TIME=10

               $RUN *APL PAR=APL,NOCONTENT,NOSUMMARY

               $RUN *APL PAR=SIZE=200000

January 1987

*APLFONTCONVERT

Contents: A program to translate VS APL transliteration code input
                into output formats and font codes appropriate for
                producing APL symbols on the Xerox 9700 page printer.

Use: APLFONTCONVERT is invoked by the $RUN command.

Program Key: *APL

Logical I/O Units Referenced:
                SCARDS - Transliteration code input to be converted into
                        APL symbols.
                SPRINT - The page-printer-oriented output from the font
                        conversion process.
                SERCOM - Error messages and summary of the conversion
                        process.

Return Codes: Always zero.

Description: The APLFONTCONVERT program reads transliteration code
                input and converts it into either portrait or landscape
                formats in the APL font to be printed on the page
                printer. All transliteration code input that is accepted
                by VS APL (*APL) is accepted by this program. Output
                lines that are too long to fit on the page width of the
                page format are wrapped around to successive lines with a
                five-column indentation.

                The transliteration code input to this program may be
                generated in *APL by first issuing the)SINK <FDname>
                command and then using *APL to output the information to
                be page printed.   Output from *APL will be directed to
                <FDname> (e.g., an MTS file) until the command)SINK
                MSINK is given.

                By default, the name of the SCARDS stream, the current
                time and date, and the user's ID are printed as a header
                on each output page. Also, the output pages are labeled
                at the bottom by means of a footer line containing the
                page number and a blank footer.

                The input stream record length is limited to 255 charac-
                ters. Long lines are truncated to this limit, with the
                line number of the first long line being printed on
                SERCOM.

                When illegal transliteration code input is encountered,
                the line number of the first line in error is printed on
                SERCOM. All illegal input characters are flagged in the

output with the ▪ square bullet character of the normal
font 1.

When the program ends, it produces a process summary of
the number of truncated long lines, the number of
erroneous input characters, and the number of input lines
processed, and indicates whether the process was com-
pleted (all input lines were processed) or terminated
(not all lines were processed, e.g., due to a user
attention or MTS I/O error).

The parameters, described below, may be used to control
the page layout. Basically, the user may select either
portrait or landscape page orientations, define footer
text, remove either the header or page numbers, and
select one-sided or two-sided output. Hence, the output
can be minimally, just APL text, and maximally, fully
labeled and annotated APL output.

The user may direct the output file to the page printer
by $RUNning *PAGEPR and specifying the output file from
*APLFONTCONVERT as input. *PAGEFONTCONVERT provides cer-
tain *PAGEPR controls (the FORMAT selection, FONTINDEX,
and either the ONESIDED or TWOSIDED parameter) as part of
its output; hence, the user may only augment these
parameters, as needed, in the PAR field of the *PAGEPR
invocation (e.g., PAR=OVERLAY=SHADING,SHIFT).

Parameters: The following parameters may be specified to direct the
conversion process. They are listed with their minimum
abbreviations being underlined. The parameters that can
be negated are also presented in their negated forms with
the "NO" prefix; however, "˜" and "-" are also accept-
able. All parameters may be separated by one or more
blanks and/or single commas.

    PORTRAIT   Produce the output in portrait page
                      orientation.

    LANDSCAPE   Produce the output in landscape page
                      orientation (the default).

    FOOTER=<str>   Define the footer to be the text contained
                      in the string <str>, which may be deli-
                      mited by either primes (') or quotes (").
                      Doubled occurrences of the delimiter
                      within <str> will be treated as a single
                      occurrences in the resultant footer
                      string. The resultant string length is
                      limited to 65 characters. The default
                      footer is "" (the null string). The
                      footer will be printed at the bottom of
                      each page, possibly with the page number,

depending on the specification of the PAGENUMBER parameter.

HEADER   Produce the standard header at the top of each page (the default).

NOHEADER   Delete the standard header from the top of each page.

PAGENUMBER   Produce the page number in the footer/page line at the bottom of each page (the default).

NOPAGENUMBER   Do not number the pages. However, if a nonblank footer was specified, print it at the bottom of each page.

TWOSIDED   Produce the output for both sides of page printer sheets (the default).

ONESIDED   Produce each output page for the the front side of a page printer sheet. In portrait mode, each page is to be treated as a front page, rather than alternating the page number and/or footer placement and justification on each page. In landscape mode, no special processing need be done.

Examples: Landscape with header and page numbers:

```
$RUN *APLFONTCONVERT SCARDS=APLFUNC SPRINT=-P
$RUN *PAGEPR SCARDS=-P
```

Portrait with header and page numbers, shifted and shaded:

```
$RUN *APLFONTCONVERT SCARDS=-L SPRINT=-P PAR=PO
$RUN *PAGEPR SCARDS=-P PAR=OVERLAY=SHADING,SHIFT
```

Portrait with header, footer and page numbers:

```
$RUN *APLFONTCONVERT SCARDS=-L SPRINT=-P PAR=PO
      FO="MAT Function in MATH Workspace"
$RUN *PAGEPR SCARDS=-P
```

Landscape with a footer but, no page numbers:

```
$RUN *APLFONTCONVERT SCARDS=-L SPRINT=-P PAR=NOPA
      FO="MAT Function in MATH Workspace"
$RUN *PAGEPR SCARDS=-P
```

Portrait with just the APL text:

```
$RUN *APLFONTCONVERT SCARDS=-L SPRINT=-P PAR=PO NOH NOPA
$RUN *PAGEPR SCARDS=-P
```

One-sided portrait with header and page numbers:

```
$RUN *APLFONTCONVERT SCARDS=-L SPRINT=-P PAR=PO,O
$RUN *PAGEPR SCARDS=-P
```

<u>*ARC</u>

Contents: The MTS implementation of the ARC file-archive inter-
                change program.

Purpose: To create and maintain file archives in a compressed form
                portable to other operating systems.

Use: The program is invoked with the $RUN command.

Program Key: *ARC

Logical I/O Units Referenced:
                SCARDS - list of files to process.
                SPRINT - comments, listings, notes, and warnings.
                SERCOM - fatal error messages.

Return Codes: 0 - Successful return.
                1 - Fatal internal error (since ARC is written in the C
                     programming language, this return code is not a
                     multiple of 4).

Description: This program is a direct port of the ARC program from
                MS-DOS and Unix. It is used to maintain archives, which
                are groups of files collected into a single file. Files
                are typically archived in this manner to ease the task of
                transferring them to other sites, particularly when using
                file-transfer protocols which do not easily allow trans-
                fer of multiple files. In addition to maintaining some
                directory information of the archived files, ARC performs
                highly sophisticated data compression algorithms on the
                archived files. While actual results will vary depending
                on the files in question, typical size reductions are:

                        Binary files   20-40%
                        Standard text files 50-60%
                        Spreadsheets   50-90%

                ARC will try a number of compression schemes, and will
                use whichever yields the smallest result. Methods   used
                are:

                (1) No compression; the file is stored as is.
                (2) Repeated-character   compression;   repeated
                     sequences of the same byte value are collapsed
                     into a three-byte code sequence. This is called
                     "packing."
                (3) Huffman squeeze; commonly occurring byte values
                     are replaced by smaller bit strings. A binary
                     tree is used to map bit strings to characters,

> and is stored along with the compressed data.
> This is called "squeezing."
>
> (4) Dynamic Lempel-Zev compression; the file is
> stored as a series of variable-sized bit codes
> representing commonly occurring character
> strings, and which are created "on the fly." The
> two variants of this method that ARC uses are
> called "crunching" and "squashing."

Note that since one of the methods involves no compres-
sion at all, the resulting archive entry will never be
larger than the original file.

By default, an archive name can be up to 8 characters,
with the string ".ARC" appended to it. Other names may
be used. For example:

```
FILENAME       would be treated as FILENAME.ARC
FILENAME.      would be treated as simply FILENAME
FILENAME.XXX would not be changed
```

Usually archives have names of the form FILENAME.ARC, in
which case only "FILENAME" need be specified in the
command to ARC.

Parameters: All commands and options are specified in the PAR field
              of the $RUN command. They may be given in either
              uppercase or lowercase. Commands and options are speci-
              fied by single letters, and a single command must be
              given. If no valid commands are present, a summary of
              valid parameters will be printed.

              One of these commands must be specified:

              A (Add)   Add listed files to archive. File lists
                            will be described in detail below.

              M (Move)   Move listed files to archive. The MTS
                            files will be destroyed after being added
                            to the archive.

              U (Update)   Update the specified files in the archive.
                            If the specified file has been modified
                            since it was last archived, the archived
                            version will be replaced by the current
                            version. Also, if the specified file does
                            not exist within the archive, it will be
                            added.

              F (Freshen)   Similar to Update, but it only replaces
                            files that already exist within the
                            archive. If a file name is specified that

                        does not exist in the archive, it will be
                        ignored.

D (Delete)   Deletes an archived file from an archive.

X or E (Extract)
                        X and E are synonyms; both request that
                        the listed files be extracted from within
                        the archive.

P (Print)   Prints the specified file onto SPRINT,
                        which may be directed to a file, to
                        PRINT, or to *SINK* (the   default).
                        Since the file is not printed with MTS
                        carriage controls, the user should extract
                        the file before printing.

L (List)   Lists the contents of the archive.   The
                        name, the original length of the file
                        before compression (in bytes), and the
                        date last modified are listed for each
                        archived file.

V (Verbose)   Verbose listing of archive contents. Sim-
                        ilar to the List command above, but also
                        shows method of compression used to store
                        the file, the time last modified, and the
                        compression ratio, as well as the current
                        size of the compressed file and the Cycl-
                        ical Redundancy Check (CRC) value of the
                        raw (uncompressed) file.

T (Test)   Test archive integrity. The CRCs for each
                        member of the archive are recomputed and
                        checked against the stored value. If any
                        values do not match, a warning message
                        will be printed. This is most often used
                        immediately after transferring an archive
                        from one site to another in order to
                        detect whether any errors occurred during
                        the transfer.

C (Convert)   Converts archived files from older storage
                        methods to current methods. This command
                        is typically used for archives that were
                        created by a version of ARC prior to
                        version 5.0. Older files are listed as
                        "crunched" in a Verbose listing, newer
                        files are listed as "Crunched" (note the
                        use of lowercase "c" versus uppercase
                        "C").

Any of the following options may also be specified.   No
spaces should separate the command and option letters.

B (Backup)   A backup of the original archive will be
                kept. An archive named FILENAME.ARC would
                have a backup named FILENAME.BAK.   By
                default, the original archive file is
                deleted.

                The normal procedure of ARC, when changing
                an archive, is to perform all changes in a
                temporary file, copying any   unchanged
                files from the original archive. When
                completed,   the   original   archive   is
                deleted, and the temporary file is renamed
                to the original archive name. With the
                Backup option, the original archive is
                renamed with the suffix .BAK, and then the
                temporary file is renamed.

I (Image)   Operate on files in "image" or binary
                mode. ARC on MTS defaults to text mode,
                where files are translated from EBCDIC to
                ASCII when being stored, and from ASCII to
                EBCDIC when being extracted. If this flag
                is specified, no translation will take
                place.   This is useful when transferring
                TeX DVI files, for example. This is also
                useful for extracting .COM and .EXE pro-
                gram files from archives before download-
                ing, in case the archive contains other
                files that are not wanted.

S (Store)   Files added to an archive will be stored
                without compression.   This saves a small
                amount of time and cost.   The archive
                contents may later be compressed with the
                Convert command.

W (Warn)   When this flag is present, no warning
                messages will be generated.   Also, ARC
                ordinarily will not allow making changes
                to damaged archives. This flag also over-
                rides that feature.

N (Note)   When this flag is present, no notes or
                comments will be generated. By default,
                ARC will display notes describing every
                phase of computation it executes while
                manipulating an archive. If both Warnings
                and Notes are disabled, ARC will not
                produce any comments or error indications.

54.4 *ARC

O (Overwrite)   Overwrite existing files when extracting.
                By default, if you attempt to extract a
                file from an archive that already exists
                on your account, ARC will ask you to
                confirm whether or not to destroy the
                existing file and replace it with the
                extracted file. If this flag is present
                the confirmation step is bypassed and
                extracted files will silently overwrite
                any existing files of the same name.

Q (Squash)   Use Squashing instead of Crunching to
                compress files. ARC uses one of four
                different methods to compress files in an
                archive.   "Crunching" is the   default,
                which is compatible with most other ARC
                programs on other systems. "Squashing" is
                newer, and often a few percent more effi-
                cient, but may not be compatible with
                other versions of ARC on other systems.

G (Garble)   Encrypt/decrypt archive entry.   This op-
                tion must be specified last. If used, it
                must be immediately followed by a pass-
                word, which will be used in encrypting the
                file.   Attempting to extract a file which
                was archived with a password will produce
                a lot of garbage and error messages,
                unless the G option and the correct pass-
                word are supplied with the Extract
                command.

For further information on the ARC program, see Computing
Center Memo 484, "ARC, The File Archiving Utility."

Examples: $RUN *ARC PAR=A MYSTUFF AFILE SOMEJUNK OTHERSTUFF

              In the above example, the files named "AFILE,"
              "SOMEJUNK," and "OTHERSTUFF" are added to the
              archive named "MYSTUFF.ARC."   If the MTS   file
              "MYSTUFF.ARC" does not exist, *ARC creates it.

              $RUN *ARC PAR=V PC1:ARC521.COM

              This example would produce a verbose listing of the
              contents of file ARC521.COM on the signon ID PC1.
              This also illustrates an archive that does not have
              the default ".ARC" ending in its name, as well as
              how to access files stored on different accounts.

```
$RUN *ARC PAR=AI -JUNK WXYZ:?.BIN ?.DVI
```

    This example will create an archive in a temporary
file, named -JUNK.ARC, and will store all files on
the signon ID WXYZ ending in ".BIN" as well as all
files whose names end in ".DVI" from the current ID.

```
$RUN *ARC PAR=X BIGFILE -?
```

    This example extracts everything in BIGFILE.ARC into
a number of temporary files. Note that, while file
names in an archive may be up to 12 characters long,
they may get truncated in this case, since temporary
files can only have up to 8-character names.

    ARC treats the "-" character as a "directory" name,
not part of the file name. Thus, the above command
extracts all the files in BIGFILE.ARC, and creates
them in a directory name "-", which is where MTS
temporary files are stored. Similarly, if the
temporary files -FILE1 and -FILE2 exist, the command

```
$RUN *ARC PAR=A TEST -?
```

    would create an archive named TEST.ARC, with FILE1
and FILE2 among its members.

```
$RUN *ARC SPRINT=*PRINT* PAR=P MYPAPERS
```

    This example sends everything in MYPAPERS.ARC to the
printer.

```
$RUN *ARC PAR=A WASTE JUNK.TXT @TRASH
```

    The list of file names given to ARC may include
indirect references. If a file name begins with an
at-sign "@", it is taken as the name of a file that
contains a list of file names. The list can consist
of any number of file names on a line separated by
blanks, and any number of lines. The list of file
names may include further indirection. If there is
no "." in the file name, a suffix of .CMD is
appended. In the above example, ARC adds JUNK.TXT
plus all files listed in the file TRASH.CMD to an
archive named WASTE.ARC. If no file is specified,
then the list is read from SCARDS.

```
$RUN *ARC PAR=A TRASH @
```

    This example reads a list of file names from the
terminal adding those files to TRASH.ARC. The list
is terminated by an end-of-file.

*ASCEBCD

Contents: The ASCII-to-EBCDIC translation program.

Purpose: To translate records from ASCII to EBCDIC code.

Use: The program is invoked by the $RUN command.

Program Key: *ASCEBCD

Logical I/O Units Referenced:
                SCARDS - ASCII records to be translated.
                SPRINT - listing of the records resulting from the
                         translation.
                SPUNCH - EBCDIC records resulting from the translation.

Return Codes: 0 - Successful return.
                4 - Invalid parameter specified.

Parameters: At most, one of the following parameters may be specified
                in the PAR field of the $RUN command.   The default
                parameter is TDAY88.

                TDAY88 Characters are translated according to the TDAY88
                        translation table.   This translation corresponds
                        to that used in MTS for translating ISO 8859/1
                        8-bit codes (ASCII) to the IBM Code Page 37 codes
                        (EBCDIC) used in MTS.

                7BIT   Characters are translated according to a 7-bit
                        translation table. This translation corresponds
                        to that used in MTS for translating the low-order
                        7 bits of ISO 8859/1 codes (ASCII) to the IBM
                        Code Page 37 codes (EBCDIC) used in MTS.

Description: Records are read from SCARDS, translated according to the
                appropriate table as specified by the parameter field,
                and written to SPUNCH.   Each output record is also
                prefixed by a blank character (for logical carriage
                control) and written to SPRINT. An end-of-file on SCARDS
                terminates the program.

                The 7BIT translation uses a 256-entry "folded" table in
                which the first and second halves are identical so that
                the effect is to ignore the high-order ASCII bit.

                See the *EBCDASC description in this volume for a program
                to translate from EBCDIC into ASCII.

          The TDAY88 translation table is given below. This table
          is also contained in the file DOC:ALLCHARTABLE.

Example: $RUN *ASCEBCD SCARDS=*T* SPUNCH=NEWFILE SPRINT=*PRINT*

          In the above example, ASCII records are read from
          the tape *T*, translated to EBCDIC, written to the
          file NEWFILE, and printed on *PRINT*.

January 1987

<u>*ASMH</u>

Contents: The MTS version of the IBM Assembler H.

Purpose: To assemble System 360/370 assembly language programs.

Use: The assembler is invoked by the $RUN command.

Program Key: *ASMH

Logical I/O Units Referenced:
                 SCARDS - the source program to be assembled.
                 SERCOM - assembler diagnostics.
                 SPRINT - listing and reference tables.
                 SPUNCH - the resulting object module.
                 0   - library of macro definitions.   If 0 is not
                          assigned, *SYSMAC will be used.
                 1   - the resulting object module.
                 2-10   - additional libraries of macro definitions.

Return Codes: 0 - Successful return.
                 n - "n" is the highest severity level of errors
                     detected.

Description: This assembler is the IBM program product Assembler H,
                 Version 2 (program number 5668-962) modified for use in
                 MTS. The language accepted by ASMH is given in the IBM
                 publication, <u>Assembler H Version 2 Application Program</u>
                 <u>ming: Language Reference</u> (form GC26-4037).   Extensions
                 to the language are discussed in the section "Assembler H
                 Extensions" of MTS Volume 14, <u>360/370 Assemblers in MTS</u>.
                 The error messages produced by ASMH are listed and
                 explained in the IBM publication, <u>Assembler H Version 2</u>
                 <u>Application Programming: Guide</u> (form SC26-4036).   Addi-
                 tions and changes to these messages are described in the
                 section "Assembler H Messages" in Volume 14.

                 The complete description of the use of this assembler is
                 given in MTS Volume 14. A brief summary of the assembler
                 options is given below.

Option	Default
ALGN or ALIGN / NOALGN or NOALIGN	ALGN
BATCH or MULT / NOBATCH or NOMULT	NOBATCH
CALIGN=n / NOCALIGN	NOCALIGN
DECK / NODECK	DECK
ESD / NOESD	ESD
EXTEN / NOEXTEN	EXTEN
FLAG(n) or MSGLEVEL(n)	FLAG(0)
LINECNT(n) or LINECOUNT(n)	LINECNT(55)
LIST / NOLIST	See Vol. 14
LOAD or OBJECT / NOLOAD or NOOBJECT	NOLOAD
MAXCREF / NOMACXREF	NOMACXREF
NUM(LEFT) or NUM / NUM(RIGHT) / NONUM	NUM(LEFT)
PEXIT=*PEXIT	
REL2 / NOREL2	NOREL2
RENT / NORENT	NORENT
RLD / NORLD	NORLD
SYSPARM(n)	See Vol. 14
TERM / NOTERM	See Vol. 14
TEST / NOTEST	NOTEST
UMAP / NOUMAP	UMAP
XREF(FULL) or XREF / XREF(SHORT) / NOXREF	XREF(FULL)

January 1987

<u>*ASMTIDY</u>

Contents: The 360/370 assembly "tidying" program.

Purpose: To edit 360/370 assembly programs into an easily readable
                 format and to indent programs containing structured
                 macros to show the program structure.

Use: The program is invoked by the $RUN command.

Program Key: *ASMTIDY

Logical I/O Units Referenced:
                 SCARDS - the input file consisting of an untidied assem-
                          bly source.
                 SPUNCH - the output file to contain the tidied assembly
                          source.
                 SPRINT - the listing of the tidied source plus MTS line
                          numbers of SPUNCH, statement numbers, and level
                          numbers.
                 SERCOM - severe error comments.

Return Codes: 0 - Successful return.
                 4 - Syntax error or line too long.
                 8 - Severe error occurred.

Description: *ASMTIDY may be used to tidy 360/370 assembler source
                 programs.   It reads an untidied assembler source program
                 from the logical I/O unit SCARDS and writes out the
                 tidied source on the unit SPUNCH. If LIST is specified,
                 *ASMTIDY will produce an indented listing of the source
                 program showing any program structure (if structured
                 programming macros are used).

                 The complete description of *ASMTIDY is given in MTS
                 Volume 14, <u>360/370 Assemblers in MTS</u>.

January 1987

<u>*ASSIST</u>

Contents: The student assembler for the IBM 360/370 assembly
               language.

Purpose: To provide a fast assembler/interpreter for student-
               written assembly language programs.

Use: The program is invoked by the $RUN command.

Program Key: *ASSIST

Logical I/O Units Referenced:
               SCARDS - assembler commands, source program, and program
                        data (the default is *SOURCE*).
               SPRINT - assembler diagnostics, source program listing,
                        and program output (the default is *SINK*).
               SERCOM - severe   error   diagnostics (the default is
                        SINK).

Return Codes: Always zero.

Description: ASSIST is a 360/370 assembler/interpreter that was writ-
               ten at Pennsylvania State University and modified to run
               under MTS by the University of British Columbia.   ASSIST
               is especially designed for use by students learning
               assembly language. ASSIST accepts a large subset of the
               G-Assembler Language provided by *ASMG, including most of
               the macro and conditional assembly features.   The
               execution-time interpreter simulates the full 370 in-
               struction set with complete error checking, meaningful
               diagnostics, and program dumps of a much smaller size
               than the normal system dumps.

               For the complete description of ASSIST, see the section
               "ASSIST Assembler and Interpreter" in MTS Volume 14,
               <u>360/370 Assemblers in MTS</u>. A brief summary of the
               assembler options is given below.

Option	Default
BATCH / NOBATCH	NOBATCH
CMPRS / NOCMPRS	NOCMPRS
COMNT / NOCOMNT	NOCOMNT
CPAGE / NOCPAGE	CPAGE
LIBMC / NOLIBMC	NOLIBMC
LIST / NOLIST	LIST
LOAD / NOLOAD	LOAD
SS / NOSS	NOSS
SSD / NOSSD	NOSSD
SSX / NOSSX	NOSSX
SIZE=n	SIZE=15
KP=n	KP=29
L=n	L=60
MACTR=n	MACTR=200
MNEST=n	MNEST=15
MSTMG=n	MSTMG=4000
NERR=n	
P=n	P=25
PAGES=n or PX=n	PAGES=10
PD=n	PD=2
T=n	T=10
TD=n	TD=0.5
TIME=n or TX=n	TIME=10
XREF=n	XREF=2

<u>*AUTOSTART</u>

Contents: The automatic batch scheduling program.

Purpose: This program enables users to automatically schedule MTS
               batch jobs that are contained in files.

Use: The program is invoked by a $RUN command.

Program Key: *AUTOSTART

Logical I/O Units Referenced:
               SCARDS - The source of commands to *AUTOSTART (defaults
                        to *SOURCE*).
               SPRINT - Output response to user requests (defaults to
                        SINK).
               SERCOM - Error messages (defaults to *MSINK*).

Return Codes: 0 - Successful return.
               8 - Error return.

Description: The *AUTOSTART program is used to automatically schedule
               MTS batch jobs that are contained in MTS files. The
               program submits the jobs through *BATCH* at times speci-
               fied by the user. After submission, the job is under the
               control of the system batch monitor and the actual time
               the job is started can be later than the submission time.
               Each time a job is submitted to run, a message will be
               sent to the user giving the job receipt number.

               Files being scheduled by *AUTOSTART differ from normal
               batch files and must conform to the following rules:

                   (1) If the file type is sequential, the first line of
                        the file must be a $SIGNON command with the CCID
                        specified ($SIGNON * will <u>not</u> work).
                   (2) If the file type is line, the $SIGNON command
                        with the CCID specified must be the first line
                        numbered 1.000 or greater.
                   (3) The signon ID on the $SIGNON command must be the
                        same as the signon ID of the user running the
                        *AUTOSTART program.
                   (4) For private files (those not prefixed with a *)
                        the owner of the file must be the user running
                        the *AUTOSTART program.
                   (5) Batch jobs submitted by *AUTOSTART do not need a
                        password after the $SIGNON line.

<u>Commands</u>

The underlined portions in the following descriptions may be used as abbreviations.

<u>AC</u>CESS

    This command displays the access that a user has to special features.

<u>DEL</u>ETE filename

    This command deletes the specified file from the automatic scheduling mechanism.

| <u>DI</u>SPLAY filename [{<u>S</u>CHEDULE|<u>O</u>UTPUT}=filename2]

    This command prints out the current scheduling information for the specified file. If the SCHEDULE or OUTPUT parameters are given, a copy of the scheduling commands is also placed in the specified file.

<u>HEL</u>P [command]

    This command prints a general description of the *AUTOSTART program or, if a specific command name is entered, a description of the specified command.

<u>MT</u>S [MTS-command]

    This command returns control to MTS command mode. Execution of the *AUTOSTART program may be resumed by the $RESTART command. If an MTS command is specified, that command will be executed upon the return to MTS command mode.

<u>QUEUE</u>

    This command lists all of the user's files currently under control of the automatic starting mechanism.

<u>RET</u>URN

    This command returns control to MTS command mode. Execution of the *AUTOSTART program may be resumed by the $RESTART command.

SCHEDULE filename [{SCHEDULE|INPUT}=filename2]

>   This command schedules the specified file or, if
>   the specified file is already scheduled, resche-
>   dules it. A full description of the scheduling
>   process is given below.

STOP

>   This command terminates the *AUTOSTART program.

$command

>   Command lines starting with $ will be interpreted
>   as MTS commands.   These commands will return
>   control to MTS command mode for the execution of
>   the specified command. An immediate return then
>   will be made to the *AUTOSTART program.

Scheduling Jobs

Issuing the SCHEDULE command will invoke the MTS editor.
The file being edited is a dummy file with a series of
commands in it that controls the frequency of the job's
submission. Different commands control different aspects
of the submission.

There are no abbreviations for any of the keywords in the
following commands.   Optional keywords are in brackets.
The periods following some of the commands are required.

Setting the Starting and Ending Dates for the Job

The following command must appear once at the front of
every run schedule.

  CONTROL FROM date1 UNTIL date2.

>   This command defines the period of time the file
>   is to be controlled by the scheduling mechanism.
>   "date1" must be at least ten minutes later than
>   the current date and time.   "date2" must be
>   greater than the date/time specified for "date1"
>   and less than the expiration date for the user's
>   signon ID. One of the following

>   {EXPIRE|EXPIRATION|EXPIRATION DATE}

>   may be specified for "date2" if the file is to be
>   controlled until the expiration of the owner's
>   signon ID. Note that if a signon ID is renewed,
>   *AUTOSTART will update any jobs that are control-
>   led by this phrase.

Defining the Action Taken During Unattended Mode

*AUTOSTART must be told what to do with the jobs during unattended mode.   One of the following three commands must appear after the CONTROL command. Tasks requiring operator intervention, such as requesting a tape mount, should not be initiated during unattended mode.

    RUN DURING UNATTENDED MODE.

        This command instructs the scheduling mechanism to run the file as it normally would even if a scheduled run time falls during unattended mode.

    SKIP RUNS DURING UNATTENDED MODE.

        This command instructs the scheduling mechanism to ignore the starting of files if the system is in unattended mode.

    DELAY RUNS DURING UNATTENDED MODE.

        This command causes a file that is scheduled during unattended mode to be delayed until MTS leaves unattended mode provided that the run window has not expired (see the description of the WINDOW command below).

Defining the Days On Which To Schedule the File

The following commands are used to define days on which the file is scheduled. The days, months, and years may be given in the following formats:

    day

        {1|2|...|30|31}

    weekday

        {SUNDAY|MONDAY|...|SATURDAY}[.]

    month

        {JANUARY|FEBRUARY|...|DECEMBER}[.]
        {1|2|...|12}

    year

        yy or 19yy

        date

            mm/dd/yy
            mm-dd-yy
            mm dd yy

Following almost all commands in this group are one or
more WINDOW commands that define the time(s) the file is
to be run on the specified days. As many RUN/WINDOW
command sequences as needed may be entered. At least one
RUN command must appear in a run schedule.

    RUN AFTER EVERY RELOAD.

        This command causes the file being scheduled to
        be started after a system reload. Note that this
        is the only command in this group that does not
        have a WINDOW command following it.

    RUN ON month day [year].
    RUN ON THE kTH DAY OF month [year].
    RUN ON THE [lTH FROM] LAST DAY OF month [year].
    RUN ON THE mTH weekday OF month [year].
    RUN ON THE [nTH FROM] LAST weekday OF month [year].

        All of these commands schedule the file to be run
        on the date specified. The abbreviations appear-
        ing above must have one the following values:

            kTH - {1ST|2ND|...|28TH}
            lTH - {2ND|3RD|...|28TH}
            mTH - {1ST|2ND|3RD|4TH}
            nTH - {2ND|3RD|4TH}

    RUN EVERY [nTH] DAY [BASED ON date].

        This command schedules the file every day or
        every "n" days. The BASED ON clause is used to
        specify the starting date for the file if it is
        different from the starting date specified in the
        CONTROL command. If nTH is specified, the BASED
        ON date is also used as the starting date for
        counting. If no BASED ON date is specified, the
        starting time on the CONTROL command is used as
        the BASED ON date. nTH must be one of the
        following values:

            {1ST|2ND|...|365TH}

RUN EVERY [nTH] {WEEK|WEEKEND|WORK-WEEK|WORK-END}
    [BASED ON date].

    This command schedules the file for the days
    falling in the given period. These periods occur
    every week or every "n" weeks. The periods are
    defined as follows:

        WEEK - Monday 00:00 am until Friday 11:59
        pm.
        WEEKEND - Saturday 00:00 am until Sunday
        11:59 pm.
        WORK-WEEK - Monday 8:00 am until Saturday
        7:59 am.
        WORK-END - Saturday 8:00 am until Monday
        7:59 am.

    The BASED ON clause is used to specify the
    starting date for the file if it is different
    from the starting date specified in the CONTROL
    command. If nTH is specified, the BASED ON date
    is also used as the starting date for counting.
    If no BASED ON date is specified, the start time
    on the CONTROL command is used as the BASED ON
    date. nTH must be one of the following values:

        {1ST|2ND|...|52ND}

RUN ON weekday OF EVERY [nTH] WEEK [BASED ON date].

    This command schedules the file for a specific
    weekday in a specific week. The BASED ON clause
    is used to specify the starting date for the file
    if it is different from the starting date speci-
    fied in the CONTROL command. If nTH is speci-
    fied, the BASED ON date is also used as the
    starting date for counting. If no BASED ON date
    is specified, the starting time on the CONTROL
    command is used as the BASED ON date. nTH must
    be one of the following values:

        {1ST|2ND|...|52ND}

RUN ON THE jTH DAY OF EVERY [nTH] MONTH [BASED ON
    date].
RUN ON THE [kTH FROM] LAST DAY OF EVERY [nTH] MONTH
    [BASED ON date].
RUN ON THE lTH weekday OF EVERY [nTH] MONTH [BASED ON
    date].
RUN ON THE [mTH FROM] LAST weekday OF EVERY [nTH]
    MONTH [BASED ON date].

These commands schedule the file for the speci-
fied day occurring in a scheduled month. The
BASED ON clause is used to specify the starting
date for the file if it is different from the
starting date specified in the CONTROL command.
If nTH is specified, the BASED ON date is also
used as the starting date for counting. If no
BASED ON date is specified, the start time on the
CONTROL command is used as the BASED ON date.
The abbreviations appearing in the above commands
must have one of the following values:

```
jTH - {1ST|2ND|...|28TH}
kTH - {2ND|3RD|...|28TH}
lTH - {1ST|2ND|3RD|4TH}
mTH - {2ND|3RD|4TH}
nTH - {1ST|2ND|...|12TH}
```

RUN ON month day YEARLY.
RUN ON THE kTH DAY OF month YEARLY.
RUN ON THE [lTH FROM] LAST DAY OF month YEARLY.
RUN ON THE mTH weekday OF month YEARLY.
RUN ON THE [nTH FROM] LAST weekday OF month YEARLY.

These commands schedule the file on the specified
day of the specified month every year in the
control interval. Note the abbreviations appear-
ing in the above commands must have one of the
following values:

```
kTH - {1ST|2ND|...|28TH}
lTH - {2ND|3RD|...|28TH}
mTH - {1ST|2ND|3RD|4TH}
nTH - {2ND|3RD|4TH}
```

Defining the Times To Schedule the File

Times may be given in the following format:

```
hh:[mm]              (24-hour clock)
hh[:mm] {am|pm}      (12-hour clock)
```

WINDOW commands, which follow the RUN commands, are used
to define the time period in which it is possible to run
the job. WINDOW commands should be given after each RUN
command, and the windows set will affect that run
schedule. The automatic scheduling mechanism will
attempt to initiate the job at the start of a run window,
if possible. If this is not possible (for example, if
MTS is not operating at the time), the job will still be
run provided it has not reached the end of the run
window. The following are formats for the WINDOW
command.

WINDOWS: start-time1[,start-time2,...]
WINDOWS: (start-time1,end-time1)[,(start-time2,
          end-time2)...]

The first form of the WINDOW command defines only
the time(s) that the job may be run; the ending
time(s) need not be specified as the program will
assume a run window of fifteen minutes. The
second form allows the user to explicitly set the
run windows. A run window has a maximum period
of 24 hours.   Thus, if the ending time of a
window is less than or equal to the associated
starting time, the ending time is assumed to
occur on the following day.   As many WINDOW
commands may be entered after a RUN command as
needed to describe all run times occurring on the
the specified day(s).

Defining the End of a Run Schedule

The following command occurs at the end of every run
schedule.

    END.

Examples: The following are examples of the commands that control
               the scheduling of jobs.

               Below is an example of a run schedule to control the
               running of a file that is to be run on the 15th and the
               last day of every month. On the 15th of the month the
               job is to be run at noon while on the last day of the
               month it is to be run at 11:59 pm. Both runs will have a
               12-hour run window.

                   Control from July 19 until expiration date.
                   Delay runs during unattended mode.
                   Run on the 15th day of every month.
                   Window: (12:00 pm,12:00 am)
                   Run on the last day of every month.
                   Window: (11:59 pm,11:59 am)
                   End.

               The next example schedules a job to be run after every
               reload.   It is run regardless of whether the system is
               unattended mode or not.

                   Control from 4pm Dec 31 until 12/31 1pm.
                   Run during unattended mode.
                   Run after every reload.
                   End.

The above schedule could have also been written as

        Control from 16:00 Dec 31 until 12/31 13.
        Run during unattended mode.
        Run after every reload.
        End.

Following is a schedule for a job that is to be run on
the first Monday, second Saturday, and last Friday day
every month. The WINDOW command will use the default
window size of 15 minutes.

        Control from Sept 1 1983 until Nov 2 1983.
        Skip runs during unattended mode.
        Run on the 1st Mon of every month.
        Window: 3am
        Run on the 2nd Saturday of every month.
        Window: 3am
        Run on the last Fri of every month.
        Window: 3am
        End.

January 1987

*BASICUM

Contents: The University of Michigan BASIC System.

                 See also *WBASIC in this volume for a description of the
                 University of Waterloo BASIC System.

Use: The BASIC system is invoked by the $RUN command.

Program Key: *BASICUM

Alt. Name: *BASIC

Logical I/O Units Referenced:

                 None

Return Codes: Always zero.

Description: This program is intended for use as a self-contained
                 system for debugging, modifying, and running programs
                 written in the BASIC language. Commands, source program
                 lines, and data are read from the MTS pseudodevice
                 SOURCE. Output from BASIC is written to the MTS
                 pseudodevice *SINK*.   The University of Michigan BASIC
                 differs from the "standard" BASIC in several minor
                 respects.   The potential user should refer to MTS Volume
                 10, BASIC in MTS.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command.   The parameters must be
                 separated by commas.   The entries may appear in any
                 order.   A standard default will be assumed for any
                 missing parameters. Following each parameter in the list
                 below is an abbreviated form for the option. The default
                 form of any option is dependent on the "device type" of
                 SINK.

                 BACKSPACE   (BS)   If the output device is recognized as
                                    a "terminal", underlining and over-
                                    striking through the use of the "back-
                                    space character" will be attempted
                                    (e.g., underlining text, such   as
                                    BASIC).
                 NOBACKSPACE (NB)   No backspacing will be attempted.

                 BATCH   (BAT) Command errors or program errors will
                                    cause BASIC to terminate operation.
                 NOBATCH   (NBT) Errors will not cause BASIC to termi-
                                    nate operation.

ECHO   (E)   All command lines will be echoed to
                    the output device.

LC   Upper- and lowercase output for system
                    messages and user-generated text will
                    be attempted.

UC   Upper- and lowercase output will be
                    disabled. <u>All</u> messages will be
                    entirely in uppercase.

NC   The user will not be queried before
                    the destroying, emptying, or freeing
                    of a BASIC file.

NOSIG   (NS)   The "signon message" after the $RUN
                    command will not be printed.

SIG   (S)   The "signon message" will be printed.

All of the above parameters except NOSIG cause the
setting of "user-defined" switches which are interrogated
by BASIC to effect the specified behavior. These
switches may be modified <u>later in the session</u> through the
use of the appropriate BASIC commands (see MTS Volume
10). The various defaults for these parameters are
detailed below according to the "device type". The PAR
field is used to override these defaults. The NOSIG
option is never defaulted and the BATCH, NC, and ECHO
options are defaulted in batch mode regardless of device
type.

Batch   <u>The device type is not checked</u>. The
                    options are NOBACKSPACE, BATCH, ECHO,
                    NC, and UC. The default output line
                    length is 131.

ARU   The options are NOBACKSPACE, NOBATCH,
                    and UC. The default output line
                    length is 255.

UMnet   The options are NOBACKSPACE, NOBATCH,
                    and LC. The default line length is
                    255.

IBM 3278   Same as UMnet.

All other devices   The options are NOBACKSPACE, NOBATCH,
                    and UC. The default output line
                    length depends on the maximum output
                    line length for the device.

| *BIBTEX

Contents: The BibTeX bibliography program for use with the LaTeX
               text-processing system.

Use: BibTeX is invoked by the $RUN command.

Program Key: *BIBTEX

Logical I/O Units Referenced:
               SERCOM - error messages, error-recovery prompting, and
                        diagnostics.
               GUSER - error-recovery prompt input.

               BiBTeX references all files explicitly by name.

Return Codes: Always zero.

Description: The complete description of the BibTeX bibliography
               program is given in the following publications:

                   LaTeX: A Document Preparation System, Leslie Lam-
                       port, Addison-Wesley Publishing Company (1986),
                       ISBN 0-201-15790-X.
                   Computing Center Memo 815, "Using LaTeX on MTS"

               BibTeX is a program for compiling a reference list for a
               document from a bibliographic database. The current MTS
               version of BibTeX, Version .98i, corresponds to the
               current LaTeX version 2.09 on MTS, but was not installed
               when LaTeX was originally made available.

               If LaTeX and BibTeX are being used on a variety of
               computing systems, note the version numbers involved,
               since .98i .bst files are incompatible with .99a and
               vice-versa. PCTeX uses .98i. The author of BibTex, Oren
               Patashnik, has announced that the program will be frozen
               at version 1.00, after which there will be bug fixes
               only; the MTS version will probably be upgraded when 1.00
               is available.

               BibTeX is run by typing

                     $RUN *BIBTEX PAR=MYFILE

               where MYFILE.TEX is the name of the LaTeX input file.
               This reads the file MYFILE.AUX, which was generated when
               Latex was run on MYFILE.TEX, and produces the file
               MYFILE.BBL. BibTeX should be run from the userID con-

| taining MYFILE.TEX (which should be the same userID from
| which LaTeX was run on that file).
|
| If the .BIB file is not on the same userID as the LaTeX
| input file, for example, if someone else's .BIB file is
| being used, then the userID must be included as part of
| the file name specified by the \BIBLIOGRPAHY command.
| For example, the LaTeX command
|
| \BIBLIOGRPAHY{1XYZ:GNUS}
|
| specifies the file GNUS.BIB residing on the userID 1XYZ.
|
| In addition to the bibliography styles described in the
| manual, there is an IEEETR style that formats entries in
| the style of the IEEE transactions. Users can also
| customize styles to their particular requirements.
| Details on how to customize styles are provided in
| Appendix A of CCMemo 815.

78.2 *BIBTEX

*CBELL

Contents: The AT&T Bell Laboratories compiler for the C programming
                 language.

| Note: It is now recommended that the *C87 C Language
| compiler be used instead of the *CBELL compiler.

Purpose: To compile C source programs.

Use: The compiler is invoked with the $RUN command.

Program Key: *CBELL

Logical I/O Units Referenced:
                 SCARDS - input of the source programs(s).
                 SPUNCH - output of the object program(s).
                 SERCOM - error messages from compiler or preprocessor.
                 2   - local include libraries used by preprocessor
                          (the standard library *CBELLINCLUDE is always
                          used).

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command. The case of the parameter
                 names is unimportant, but case distinction is made in the
                 values of the parameters for DEFINE and UNDEFINE.

                 ASMSOURCE / NOASMSOURCE   Default: NOASMSOURCE

                     With this option, the assembler source produced by
                     the compiler is made available in the temporary file
                     -__#C.ASM.

                 ASSEMBLE / NOASSEMBLE   Default: ASSEMBLE

                     NOASSEMBLE suppresses the assembly phase.

                 BREG=n   Default: BREG=1

                     The BREG option is used to set the number of base
                     registers used to address the source code.   This
                     compiler is less sophisticated than many and is
                     unable to determine how many base registers are
                     needed to address its code. An addressability error
                     in the assembly phase indicates the need for addi-
                     tional base registers. The maximum number of base
                     registers that can be specified is 5. The number of
                     registers reserved for use as bases should be kept
                     as small as possible since increasing the number of
                     base registers decreases the number of work regis-

                     ters. The amount of space used for local variables,
temporaries, and copies of the parameters is limited
to 4K bytes (1 page). There is no feature in this
compiler to use more than one base register for
accessing this local data.

     CSOURCE / NOCSOURCE   Default: NOCSOURCE

        With this option, the output from the preprocessor
is made available in the temporary file, -__#C.CMP.

     COMMENTS / NOCOMMENTS   Default: NOCOMMENTS

        COMMENTS causes the preprocessor phase to pass
source comments through into its output.

     COMPILE / NOCOMPILE   Default: COMPILE

        NOCOMPILE suppresses the compiler and assembly
phases.

     DEFINE(name=value) / DEFINE(name)   Default: None

        DEFINE(name=value) causes the preprocessor to gener-
ate the equivalent #define statement. If the value
is omitted, the value 1 is used.

     PRETDAY / TDAY88   Default: TDAY88

        PRETDAY assumes that the program source file con-
tains pre-T-Day translation codes (i.e., the file
was created before February 22, 1988 and not trans-
lated). See Computing Center Memo 480, "Translation
Day Character Code Changes on MTS."

     RENT / NORENT   Default: NORENT

        RENT causes the assembler phase to test for reen-
trancy violations (stores within a CSECT). This
feature is provided primarily for writing programs
to be incorporated into the operating system and
would not normally be used for an applications
program.

     STACK_CHECK / NOSTACK_CHECK   Default: STACK_CHECK

        The current version of *CBELL storage protects the
first page after the end of the allocated stack so
that references into this region will result in a
protection exception. This is done regardless of
the setting of this parameter. Future versions may
generate code to test for stack overflow if the
STACK_CHECK option is specified.

TEST / NOTEST   Default: TEST

    NOTEST   suppresses the assembler-phase generation of
    SYM records, which can be used by the SDS debugger.

UNDEFINE(name)   Default: None

    UNDEFINE causes a "#undefine name" to be processed
    by *CBELL.

Return Codes: 0 - Successful return.
              4 - Error in PAR field, a file needed by *CBELL does not
                  exist, or insufficient access to the file.
              8 - Error message from compiler.
            12 - Insufficent memory for compilation.
            16 - Internal error in compiler (should not occur).

Description: *CBELL is a program that successively links together the
               three phases of the compiler, where the output of one
               phase is transmitted to the next in a temporary file.
               The three phases are:

          (1) The preprocessor, which processes entities such
               as #define and #include statements.
          (2) The main compiler, which turns the preprocessed C
               source into assembler-language source.
          (3) The *ASMH assembler, which turns the assembler-
               language output of the compiler into object code.

               The following standard include files are supported:
               <assert.h>, <ctype.h>, <errno.h>, <limits.h>, <math.h>,
               <setjmp.h>, <stdarg.h>, <stdio.h>, <stdlib.h>, <time.h>,
               and <times.h>. In addition, several nonstandard files
               are supported: <mtsio.h> and <mts.h>.

               Object programs produced by *CBELL must be run using the
               library in *CBELLLIB. This library supports some Berke-
               ley Unix 4.2 system routines to enhance portability.

               The C language is described in numerous texts, one of the
               most widely used being The C Programming Language, by
               Brian Kernighan and Dennis Ritchie. The use of C in MTS
               is described in Computing Center Memo 472, "*CBELL - Bell
               Labs C in MTS."

Example: $RUN *CBELL SCARDS=SOU SPUNCH=OBJ
               $RUN OBJ+*CBELLLIB SCARDS=INFILE

January 1987

*CCBIBLIOGRAPHY

Contents: A bibliographical list of current documentation for MTS.

Use: The $COPY command should be used, e.g.,

                    $COPY *CCBIBLIOGRAPHY

Description: The list of available documentation is organized by
               subject category.

*CCDELIVERY

Contents: A schedule of delivery times for courier service between
               the Computing Center and the Main Campus public batch
               stations.   The following items are currently handled by
               this service:

                    Page printer output printed at CNTR
                    Plotter output plotted at CNTR
                    Magnetic tapes both to and from CNTR
                    Floppy disks both to and from CNTR
                    Phototypesetter output set at CNTR

Use: The $COPY command should be used, e.g.,

                    $COPY *CCDELIVERY

January 1987

*CCHOURS

Contents: A list of the hours of the Computing Center and its
               public stations on campus.

Use: The $COPY command should be used, e.g.,

                   $COPY *CCHOURS

*CCPHONES

Contents: A list of Computing Center telephone numbers.

Use: The list may be obtained via the $COPY command, e.g.,

                   $COPY *CCPHONES

*CCPUBLICATIONS

Contents: A list of the current Computing Center Memos, Quicknotes,
               MTS Volumes, and other publications.

Use: The $COPY command should be used, e.g.,

                   $COPY *CCPUBLICATIONS

January 1987

<div align="center">

*CCQUEUE
_____
</div>

Contents: The queueing program for requesting postprocessing of
                user plot descriptions.

Purpose: To place the user's plotting request(s) on the queue for
                plotting.

Use: This program is invoked by the $RUN command.

Program Key: *CCQUEUE

Logical I/O Units Referenced:
                SCARDS - queueing requests.
                SPRINT - plot receipt numbers and prompting messages for
                         queueing requests.
                SERCOM - error comments.
                GUSER - user responses to error comments.

Return Codes: Always zero.

Parameters: A single queueing request may be specified in the PAR
                field of the $RUN command, if no requests are to be read
                from SCARDS.

Description: The complete description of the plotting system is given
                in MTS Volume 11, Plot Description System.

                Each queueing request specifies a file or tape containing
                one or more plot descriptions. *CCQUEUE reads the plot
                descriptions, determines the cost of plotting, places the
                request in the system plotting queue, and charges the
                user for the plotting costs. At each scheduled plotting
                time, the system postprocessor removes the request from
                the queue, reads the user's file (or tape), and produces
                commands for the plotter.

                *CCQUEUE prompts for queueing requests using the message
                "ENTER PLOT REQUEST".   The response may be a queueing
                request for a file or tape, a line consisting of the
                string "MTS", a line beginning with a "$", or an
                end-of-file indicator. If the queueing request is for a
                file, the request should consist of the file name,
                optionally followed by one or more blanks and a scale
                factor.   Explicit concatenations are permitted, as are
                line number ranges. Only permanent files may be queued--
                temporary files may not be queued.

                The response "MTS" causes a return to MTS command mode.
                A response that begins with a dollar sign is treated as

an MTS command that is to be executed immediately. For either response, "MTS" or an MTS command, execution of *CCQUEUE may be resumed by entering a $RESTART command (unless an MTS command causes *CCQUEUE to be unloaded). An end-of-file response terminates execution of *CCQUEUE.

If the request is for a magnetic tape, the tape should already be mounted. The request should contain first the pseudodevice name, followed by the tape ID. Parameters must be separated by one or more blanks and/or commas. There are three optional parameters. One is the scale factor, which may appear anywhere after the pseudodevice name. Another is the FILES parameter, which may also appear anywhere after the pseudodevice name. This takes the form FILES=n, where "n" is the number of files to be plotted. The default is FILES=1. Finally, the POSN parameter may appear anywhere after the tape ID. This takes the form, POSN=string, where "string" specifies a file on the tape (any string legal for the POSN control command). If this parameter is not specified, *CCQUEUE begins reading at the current tape position (even if that is in the middle of a file). (At postprocessing, the tape will be mounted and positioned appropriately, even to the middle of a file.) *CCQUEUE and the postprocessor use whatever blocking is in effect at the time the queueing request is entered. *CCQUEUE does not rewind the tape after reading it. Pool tapes may not be queued.

Any plot request may include a scale factor. The scale factor will be applied to each point of the plot. This parameter takes the form SCALE=x, where "x" is a positive integer, F-type, or E-type real number.

Users can request delivery of plots to another station by specifying the DELIVERY=station parameter on the queue request, e.g., DELIVERY=NUBS. The DELIVERY parameter is effective for only one queue request. If the DELIVERY parameter is not specified, the setting of the MTS $SET DELIVERY option will be used (which defaults to the Computing Center).

Users can request "quick" plotting by specifying the ASAP parameter on the queue request. This will provide a faster than normal service. Ballpoint pens will be used instead of liquid ink and the appearance of the plot will differ; lines will be thinner and not as dark, and dots may not be drawn very well. The plotter will use only red, blue, and black ballpoint pens, and pen size changes will not be allowed. See the public file *PLOTTIMES for the schedule of ASAP plotting service.

Error Processing:

If *CCQUEUE encounters an error in the user's plot
description file, it prints a comment describing the
error, together with the file name and line number where
the error was discovered. If an error is encountered in
batch mode, reading of the current plot description
stops, the plot is not queued, and *CCQUEUE begins
processing the next request, if any. In conversational
mode, the user is asked if he wishes *CCQUEUE to
continue. The legal responses are YES (or OK) to
continue processing (taking the error recovery action
described below), and NO, to stop processing at this
point (this does not abort the plot). The user may also
enter MTS or an MTS command preceded by a "$", in which
case control will be returned to MTS. (The user may
restart using $RES, and he will again be prompted for a
response.) The error recovery actions are

Error	Action Taken by *CCQUEUE
Record not a plot record	Record ignored
Actual record length not equal to expected length	Record ignored
Missing PBGN	PBGN (logically) inserted with default normalizing factor
Invalid PSYM record	Record ignored
Invalid PPEN record	Record ignored
Missing PEND	PEND (logically) inserted

If the user indicates that processing should continue
after an error is encountered, and then queues the plot,
the same error recovery action will be taken by the
postprocessor.

A line beginning $CONTINUE WITH ... will be treated as
an invalid plot line. Implicit concatenation may not be
used in plot description files (or tapes).

Queueing:

After *CCQUEUE finishes reading the plot description
(because it has reached the end or the user has told it
not to continue after an error), it prints the number of
plots (blocks), the plotting time required, the amount of
paper required, and the cost. In conversational mode,
the user is asked whether the plot should actually be
queued. If he replies NO, the plot is aborted and he is

prompted for a new plot request (unless the plot request
was taken from the PAR field). If he replies YES, or if
*CCQUEUE is being executed in batch mode, the plot
request is placed in the queue, he is charged for the
plot, and a receipt number is printed (this receipt
number must be used to obtain the plot at the output
window later). In addition, if the plot description was
contained in one or more disk files, *CCQUEUE will, if
necessary, permit the queued files so that the postpro-
cessor, which is a program with PKEY=*CCQUEUE, may access
the plot descriptions. *CCQUEUE will establish READ
access for the file to the postprocessor if necessary.
(Specifically, the file will be permitted "READ PKEY=*
CCQUEUE" unless the postprocessor already has read
access. See the description of the $PERMIT command in
MTS Volume 1, <u>Michigan Terminal System</u>.) A message is
printed for each file so permitted. If *CCQUEUE cannot
determine the permit status of a file, a message is
printed, and <u>the user must be sure that the file is
permitted before the plot request is postprocessed</u>. See
the section below on postprocessing.

The plot will be aborted if it is null, if the user's
plotting time limit (local, global, or ID) has been
exceeded, if the user indicated abort after an attention
interrupt (see below), or if reading stopped on an error
while in batch mode.

<u>Cancellation</u>:

A queued plot may be canceled by entering CANCEL nnnnnn
where "nnnnnn" is the receipt number, whenever *CCQUEUE
is prompting for a plot request. A plot can be canceled
only by the signon ID which queued it. If the plot
specified is still in the queue (i.e., it has not been
postprocessed), it will be canceled and a message to that
effect will be printed. Plotting charges will be rebated
automatically at a later time.

<u>Attention Interrupt Processing</u>:

If the user issues an attention interrupt, processing of
the current plot request is suspended; the subsequent
action taken by *CCQUEUE is as follows: (1) If it was
parsing a plot request, the plot request is discarded and
the user is prompted to enter a new request; (2) *CCQUEUE
will print "READING:" or "QUEUEING:", depending upon
whether it was in the process of reading the plot
description or queueing the plot. In either case, if the
user enters a null line or an end-of-file, *CCQUEUE will
resume processing the current request. If the user
enters "MTS" or an MTS command beginning with a "$",
*CCQUEUE will return to MTS command mode; if a command

was given, it will be immediately executed by MTS.   If
the user enters anything else, the plot request will be
aborted.

If the user issues a second attention interrupt while
*CCQUEUE is processing the first interrupt, an immediate
return is made to MTS command mode.   The user may
subsequently reenter *CCQUEUE via the $RESTART command.

<u>Postprocessing</u>:

After a plot request has been queued, the user's plot
description must be available for reading by the system
postprocessor at the scheduled plotting time. If the
plot description is in a file, the file must be per-
mitted, and it must not be locked at any level higher
than READ. If the file is not permitted, the plot will
be canceled; if it is locked, the plot request will
remain queued until the next scheduled plot time at
whichit is accessible. If the plot description is on a
tape, the tape must be available for mounting (e.g., not
already mounted); otherwise, the plot will remain queued
until the tape is available at a scheduled postprocessing
time. If the tape ID is incorrect, the plot will be
canceled.

Examples: In the following examples, terminal output appears in
               uppercase and user input appears in lowercase.

               In the first example, the plots contained in two files,
               PLOTS and MOREPLOTS, are to be queued. At line 2 in
               MOREPLOTS, *CCQUEUE discovers that a PBGN record seems to
               be missing, and asks the user if it may assume a PBGN
               record with a default normalizing factor. The user OKs
               this, so *CCQUEUE continues reading the plot file.   At
               line 2.5, *CCQUEUE discovers a PPEN record with an
               invalid color, and asks the user whether it may ignore
               the line.   The user invokes the file editor to examine
               the line, and then tells *CCQUEUE not to continue.   The
               user then OKs queueing of all plots up to line 2.5 of
               MOREPLOTS.

```
  #$r *ccqueue
  EXECUTION BEGINS
   .
   .
  ENTER PLOT REQUEST:
  plotfile
    2 PLOTS; PLOTTING REQUIRES  94 SEC. AND 19 IN.; $.31
    PEN WAS UP 34% OF THE TIME
  OK?
  ok
  PLOT ASSIGNED RECEIPT # 516061
```

```
                  ENTER PLOT REQUEST:
                  plots+moreplots
                  MISSING PBGN RECORD.
                  DISCOVERED AT LINE      2.000 IN MOREPLOTS
                  OK TO USE DEFAULT?
                  y
                  INVALID PEN TYPE.
                  DISCOVERED AT LINE      2.500 IN MOREPLOTS
                  OK TO IGNORE?
                  $ed moreplots
                 #$ED MOREPLOTS
                 :p 2.5
                 :     2.5    PPEN??PINK
                 :stop
                  OK TO IGNORE?
                  n
                  PLOT UP TO LAST ERROR CAN BE QUEUED.
                    2 PLOTS; PLOTTING REQUIRES  47 SEC. AND 9 IN.; $.17
                    PEN WAS UP 41% OF THE TIME
                  OK?
                  ok
                  PLOT ASSIGNED RECEIPT # 516064.
                  .
                  .
```

In the second example, the user does not have appropriate
access to XXXX:PLOTS for *CCQUEUE to determine its permit
status. Note that the user must make sure that the file
is permitted before the next postprocessing time.

```
    #$r *ccqueue
     EXECUTION BEGINS
     .
     .
     ENTER PLOT REQUEST:
     xxxx:plots
       1 PLOTS; PLOTTING REQUIRES 35 SEC. AND 7 IN.; $.13
       PEN WAS UP 61% OF THE TIME
     OK?
     y
     **PERMIT STATUS OF "XXXX:PLOTS       " UNKNOWN.
     PLOT ASSIGNED RECEIPT # 516068.
     .
     .
```

January 1987

<u>*CCSOFTWARE</u>

Contents: A list of software supported by the Computing Center.

Use: The $COPY command should be used, e.g.,

                    $COPY *CCSOFTWARE

January 1987

<u>*CDUPDATE</u>

Contents: The "context-directed" update program.

Use: The program is invoked by the $RUN command.

Program Key: *CDUPDATE

Logical I/O Units Referenced:
                 SCARDS - update records to be applied to the master file.
                 SPRINT - source and command listings and error messages.
                 SERCOM - error messages and commands containing errors
                          (if assignment different from SPRINT).
                 0   - old master file (input).
                 1   - new master file for updated source (output).

Parameters: The PAR field of the $RUN command may be used to specify
                 any of the options that may be selected by the /OPTIONS
                 command.

Return Codes: 0 - Successful return.
                 4 - Errors detected while preprocessing the input and
                      the update phase was skipped.
                 8 - Errors detected while applying the update.
                 12 - The program ran out of storage (this might result
                      from a /BUFFER command that was not correctly
                      terminated). The program will terminate processing,
                      but will attempt to produce the command listing that
                      will usually indicate the command causing   the
                      problem.
                 16 - Internal error in CDUPDATE occurred. The program
                      will terminate processing, but will attempt to
                      produce the command listing.

Description: CDUPDATE is a program that applies updates to an "old
                 master file" to produce a new "updated master file."   It
                 is typically used in maintaining a program in the form of
                 a "base-level source" (the master) and a set of updates
                 to be applied to it to produce the current version.

                 The program has the following properties:

                 (1) Like *UPDATE, it takes an old master file and a
                      control file (consisting of update commands and
                      insertions) and produces updated source as an
                      output file, without changing the original.
                 (2) Records are located contextually using patterns
                      to describe the records to be found, rather than
                      by sequence numbers. The patterns allowed are a
                      subset of SNOBOL4 patterns, and include those

provided by the MTS file editor.
(3) It is possible to move large sections of text either forward or backward in the output file, relative to the position in the input file.
(4) The program will automatically add a "change-code" to inserted or modified records.
(5) The program will apply multiple updates in a single pass through the input file.

The input to this program consists of a sequence of _independent_ update sets, each of which must be delimited by the commands /BEGIN and /END. Only comments and /OPTIONS commands may appear outside of these delimiting commands. The /BEGIN record may also specify an update name and/or change code, which will be placed in each record inserted by that update. Thus, the input for one update set looks like:

```
/BEGIN update-name
 ... (other update commands and insertions)
/END
```

Logically, the separate updates are applied sequentially. That is, each update is applied to the output file produced by the previous update. In fact, the whole series of updates is applied in a single pass through the old master file.

Each update level takes a queue of input records and produces a queue of output records. The output from one level is the input to the next. The input to the first is, of course, the input file, and the output from the last is written to the output file. The whole process is performed incrementally, so that it is not necessary to have large numbers of records in memory at any time.

This multilevel facility allows the user to group together all changes to the source which relate to a particular functional change in the program. The change-code can be used to identify the resulting changes in the new source.

The complete description of the CDUPDATE program is given in Computing Center Memo 434, "The CDUPDATE Program." This memo includes a description of the following:

(1) Commands and options.
(2) CDUPDATE listings.
(3) Change-code processing.
(4) CDUPDATE buffers.
(5) CDUPDATE patterns.
(6) Examples.

January 1987

Commands: A list of the commands is given below. The descriptions
                 of the commands are given in Computing Center Memo 434.

                      /BEGIN [update-name] [CHANGECODE=string]
                      /BUFFER name [{INPUT|OUTPUT}] [operands]
                      /BUFFER END
                      /COMMENT anything
                      /.anything
                      /COPY operand [/operand] ...
                      /DELETE [MAXCOUNT=n] pattern [TO operand]
                      /DEBUG debugoptions
                      /EDIT [HOLD] replacement [/replacement] ...
                      /END [update-name]
                      /EOF
                      /INCLUDE name [{INPUT|OUTPUT}]
                      /MESSAGE anything
                      /OPTIONS optionlist
                      /SKIP operand [/operand] ...

Options: Several CDUPDATE options may be selected, either via the
                 PAR field of the $RUN command, or by using a /OPTIONS
                 command. The options are listed below. The descriptions
                 of the options are given in Computing Center Memo 434.

                      ANCHOR={ON|OFF}
                      {ANYCASE|AC}={ON|OFF}
                      {CHANGECODE|CC}=changecodeoptions
                      COMMANDLIST={ON|OFF}
                      CLIST={ON|OFF}
                      LIST=listoption
                      LIST=(listoption1,listoption2)
                      {SOURCEMARGINS|SM}={n|n-m|FREE}
                      TITLE="string"
                      VERSION=n.n

January 1987

<u>*CKID</u>

Contents: A Computing Center ID (signon ID) checking program.

Purpose: To provide additional security for signon IDs that must
                 be shared by several users.

Use: The program is invoked using the $RUN command.

Program Key: *CKID

Logical I/O Units Referenced:
        0   - the authorization file that lists the group
                   signon IDs that CKID is being used to protect
                   together with the individual signon IDs allowed
                   to use them.
        1   - the log-file that will be used to hold informa-
                   tion about each successful or unsuccessful sign-
                   on attempt.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command:

                 ID_TRIES=n

                     The ID_TRIES parameter specifies the number of
                     attempts to be given to the user to enter an
                     individual signon ID that is authorized to use the
                     group signon ID. "n" may be 1, 2, or 3; the default
                     is 1.

                 PW_TRIES=n

                     The PW_TRIES parameter specifies the number of
                     attempts to be given to the user to enter the
                     correct password for the individual signon ID.   "n"
                     may be 1, 2, or 3; the default is 1.

                 SIGFILE={<u>ON</u>|OFF|ALWAYS}

                     The SIGFILE parameter controls the processing of the
                     user's sigfile. If ON is specified (the default),
                     the user's sigfile is processed unless the user
                     specifies SIGFILE=OFF on one of the input lines. If
                     OFF is specified, the user's sigfile is not pro-
                     cessed unless the user specifies SIGFILE=ON on one
                     of the input lines. If ALWAYS is specified, user
                     sigfile-processing is done regardless of any SIGFILE
                     specification by the user.   Note that when user-
                     sigfile processing is enabled, the $RUN *CKID com-

mand should be the last command in the main sigfile
since control will not be returned to the sigfile
after processing of the user's sigfile is completed.

TEST

If the TEST parameter is specified, CKID will not
check the password of the individual signon ID, will
not issue the $SOURCE command for the user's sig-
file, will not set SIGFILEATTN=OFF, and will not
sign the user off if the CKID checking fails. This
parameter is intended for testing authorization
files for correctness of syntax.

Return Codes: Always zero.

Description: In the description that follows, the term "group signon
               ID" refers to the signon ID(s) that are to be shared,
               while the term "individual signon ID" refers to some
               other signon ID. To successfully sign on to an account
               protected by CKID, a user must know the group signon ID
               and its password together with an individual signon ID
               that has been specifically authorized to use the group
               signon ID and the password associated with the individual
               signon ID.

               CKID reads from *MSOURCE* using the prefix 'ID?'   to
               prompt for the individual signon ID and the prefix 'PW?'
               to prompt for the password associated with the individual
               signon ID. When CKID is run from a sigfile (as is
               intended), it will unconditionally set SIGFILEATTN of
               OFF.

               If the individual signon ID entered is authorized to use
               the group signon ID and the individual-signon ID/password
               combination is correct, the SIGFILE of the individual
               signon ID (if any) is processed and an entry is made in
               the log-file assigned to logical I/O unit one. If the
               individual signon ID is not authorized to use the group
               signon ID, the individual signon ID is invalid, or the
               individual signon ID/password combination is incorrect,
               an entry is placed in the log-file, an explanatory
               message is printed on *MSINK*, and the user is signed
               off.   CKID never causes the project SIGFILE associated
               with the individual signon ID (if any) to be processed.

               The individual signon ID and password may be entered on
               the same line separated by one or more blanks. In
               addition, the string, "SIGFILE=OFF", may be entered
               anywhere on an input line to suppress the individual
               signon ID SIGFILE processing (subject to the setting of
               the SIGFILE parameter).

The authorization information in the file assigned to logical I/O unit 0 consists of one or more lines in one of the following two formats:

    (1) NAMES exp1 exp2 ...

        where "exp" is an expression of the form

           ([ID=]ccid,[NAME=]'name')

    (2) group-ID exp1 exp2 ...

        where "group-ID" is a group signon ID that is being shared and "exp" is either an individual signon ID being granted access to the group signon ID, or an expression of the form

           ([ID=]ccid,[NAME=]'name')

        or

           ([ID=]ccid,*)

        If the first expression form is given, CKID will attempt to set the specified name using the password of the owner ID (the name must have been previously enrolled in *USERDIRECTORY). This is equivalent to issuing the $SET NAME='name' command for the group ID. If the second expression form is given, CKID assumes that the name for the specified signon ID was given on a NAMES record in the authorization file (see (1) above). This form allows names to appear only once in the authorization file.

The simplest example of an authorization file is as follows where there is one group ID per line followed by the single individual IDs:

    WABC W001 W003
    KSTA KSTB W003

A more complex example is as follows in which CKID will attempt to set the specified name for the group ID to the individual name associated with the individual signon ID:

    WABC (W001,'Mary Smith) (W003,'John Brown')
    KSTA (KSTB,'Paula Green') (W003,'John Brown')

The most complex example is as follows in which a NAMES line is added to the authorization file to lessen the number of times an individual name is repeated:

```
        NAMES (W003,'John Brown')

        WABC (W001,'Mary Smith) (W003,*)
        KSTA (KSTB,'Paula Green') (W003,*)
```

The authorization file must be permitted at least READ to each group signon ID and/or PKEY=*CKID.

After successful signons, the line written to the log-file contains the following information:

```
        group signon ID
        individual signon ID
        *MSOURCE* device-type
        time
        date
        answerback
```

The line written to the log-file after unsuccessful signons contains the same information listed above except that the answerback is replaced with the string '*FAIL*' followed by one of the following reasons:

File Access   one of the files used by CKID was inac-
                cessible (should not occur unless the
                SIGFILE was set up incorrectly).

ID Access   the individual signon ID is not allowed to
                access this group signon ID.

Password   the password entered doesn't match the
                actual password associated with the indi-
                vidual signon ID.

No ID/PW   no individual signon ID and/or password
                was given or an invalid individual signon
                ID was entered.

The log-file must be permitted at least WRITE-EXPAND (WE) to each group signon ID and/or PKEY=*CKID. Extra security is provided if the log-file is not stored under any of the group IDs and is not permitted WRITE-CHANGE (WC) to any of them.

```
Example:        $SIGNON SBCD
                 ...
                $CREATE SIG
                $CREATE LOGFILE
                $CREATE AUTHFILE
                $PERMIT SIG R K100
                $PERMIT LOGFILE WE PKEY=*CKID
                $PERMIT AUTHFILE R PKEY=*CKID
                $COPY *SOURCE* SIG
                $SET SIGFILEATTN=OFF
                $RUN *CKID 0=SBCD:AUTHFILE 1=SBCD:LOGFILE T=0.25
                $ENDFILE
                $COPY *SOURCE* AUTHFILE
                SBCD SB01 SB02 SB03 SCAB
                K100 SB01 SB03 K123
                $ENDFILE
```

The above lines could be used to establish the files
necessary to protect the group signon IDs SBCD and
K100. The signon IDs SB01, SB02, SB03, and SCAB are
authorized to use SBCD, while the signon IDs SB01,
SB03, and K123 are authorized to use K100.

```
$SET SIGFILE=SBCD:SIG SIGFILEATTN=OFF
```

This command must be issued twice, once while signed
on under the ID SBCD and again while signed on under
K100. It will establish the file, SBCD:SIG, as the
SIGFILE for both group signon IDs and disable
attention interrupts while the SIGFILE is being
processed.

```
$SIGNON SBCD
<password for SBCD>
SB02
<password for SB02>
```

        or
```
$SIGNON SBCD S
<password for SBCD>
SB02 <password for SB02>
```

        or
```
$SIGNON SBCD S
<password for SBCD>
SIGFILE=OFF SB02 <password for SB02>
```

In the three examples shown above a user signs on to
the group signon ID, SBCD, and enters the individual
signon ID, SB02, and its associated password in
response to prompts from *CKID. In the final
example no processing of the SIGFILE associated with
the individual signon ID, SB02, takes place.

January 1987

*CKSIG

Contents: The program to validate a sign-on by checking specified values against the day of the week, a GUINFO item, a SNS item, the time of day, or the signon ID.

Use: The program is invoked by the $RUN command.

Program Key: *CKSIG

Logical I/O Units Referenced:
      SCARDS - the specification of the values to be checked.
      SPUNCH - if specified, a line file into which is placed the signon ID, the time of the sign-on, and whether the sign-on succeeded or failed as determined by CKSIG.
      SERCOM - error comments and TEST and TRACE output.

Parameters: The following parameters may be specified in the input stream to SCARDS. One or more parameters may be entered on each input line; blanks or commas may be used as separators. The underlined portion of each parameter may be used as an abbreviation.

      DAY   If specified, the current day of the week must be one of the specified days. The day is specified in the form DAY=day or DAY=(day ...). The minimum abbreviations for the days are SU, M, TU, W, TH, F, and SA.

      GUINFO   If specified, the current information returned by GUINFO must meet the condition specified for the value. The GUINFO information is specified in the form

         GUINFO=(specification) or
         GUINFO=((specification)[,(specification)]...)

         where "specification" is

         ITEM={itemno|itemname} [LENGTH=length]
         [OFFSET=offset] [VALUE{<|<=|=|>=|>|˜=}]
         {integer|"string"|'string'|string}

      itemno   is the number of a GUINFO item.
      itemname is the name of a GUINFO item.
      length   is the number of bytes or characters to be included in the comparison. Padding on the right with blanks is provided, if necessary. If not specified, a default

of 4 bytes is used if the specified
value is an integer, or the length of
the specified string is used if the
value is a string.

offset   is the number of bytes or characters in
the information returned by GUINFO to be
skipped before starting the comparison.
If not specified, a default value of 0
is used.

integer is an integer consisting of an optional
sign and digits. This integer is con-
verted into a 4-byte, binary number
before comparing it with the current
GUINFO value.

string   is a string of characters to be compared
with the current GUINFO value. If the
string contains all digits, digits pre-
ceded by a plus or minus sign, or a
comma or blank, it must be enclosed in
primes (') or quotation marks ("); if
the string is thus enclosed, then two
such marks must be used to represent a
single prime or quotation mark within
the string.

ID   If specified, the current signon ID must be one
of the specified signon IDs. The signon ID is
specified in the form ID=ccid or ID=(ccid ...).

SNS   If specified, the SNS information for *MSOURCE*
must meet the condition specified for the value.
The sense information is specified in the form

    SNS=(specification) or
    SNS=((specification)[,(specification)]...)

where "specification" is

    [LENGTH=length] OFFSET=offset
    [VALUE{<|<=|=|>=|>|˜=}]
    {"string"|'string'|string}

length   is the number of bytes or characters to
be included in the comparison. Padding
on the right with blanks is provided, if
necessary. If not specified, the de-
fault of the length of the string speci-
fied for the value is used.

offset   is the number of bytes or characters in
the information returned by the SNS
control command to be skipped before
starting the comparison.

string   is a string of characters to be compare-

dwith the current SNS information. If the string contains a comma or blank, it must be enclosed in primes (') or quotation marks ("); if the string is thus enclosed, then two such marks must be used to represent a single prime or quotation mark within the string.

TIME   If specified, the current sign-on time must meet the specified condition. The current sign-on time is specified in the form

      TIME{<|<=|>=|>}hh[:mm[:ss]]

If ":mm" or ":ss" are not specified, the default value of :00 is used.

END   This parameter is used to terminate a keyword parameter group.

The following parameters may be specified in the PAR field of the $RUN command. The underlined portion of the parameter may be used as an abbreviation.

TEST   If specified, a failure does not produce a sign-off; instead, a success or failure message is written on SERCOM.

TRACE   If specified, the values used in each test and the results of the test are written on SERCOM. This is useful for debugging a set of parameters.

Return Codes: None.

Description: The values specified in the keyword parameters are compared with the current value. If any condition for a parameter is true, that parameter is considered to be successful. If any parameter is not successful, the user is signed off when control is returned to MTS command mode.

Parameters may be grouped by terminating each group with the END parameter (or the end of input on SCARDS). In the case of groups, a sign-off is produced on return to MTS command mode only if at least one parameter in each group was not successful.

For further information about the GUINFO information, see the GUINFO subroutine description in MTS Volume 3, System Subroutine Descriptions. For further information about the SNS information for a particular I/O device, see the appropriate section of MTS Volume 4, Terminals and Networks in MTS.

Example: $RUN *CKSIG
                  DAY=(M,TU,W,TH,F) TIME>=08 TIME<=16:30
                  ID=T012
                  GUINFO=((ITEM=SYSOLOAD 0) (ITEM=LSS 0))
                  END
                  ID=T987
                  GUINFO=(ITEM=UNATMODE 0)

                       For the sign-on to continue in the above example,
                       the signon ID must be T012, the day of the week must
                       be Monday through Friday, the sign-on time must be
                       between 8 am and 4:30 pm, and either the system
                       overload indicators must be off or LSS must be off;
                       or the signon ID must be T987 and the system must
                       not be running in unattended mode.

January 1987

*CLPARSEGEN, *CLPARSER

Contents: A command-language syntactic parser.

Use: The *CLPARSEGEN program is invoked by the $RUN command to
               produce a syntax table which is used in conjunction with
               the user's compiled program.

Program Key: *CLPARSEGEN (for *CLPARSEGEN)
               *EXEC (for *CLPARSER)

Return Codes: 0 - Successful return.
               4 - Error return.

Description: Many programs dynamically interact with users, reading
               and interpreting commands the user   enters   on   the PAR
               field of the $RUN command or in response to prompts from
               the program. Other programs must perform some   sort   of
               syntactic analysis of data. Writing procedures to ana-
               lyze the syntax of these commands or data can be   time
               consuming and awkward. The KWSCAN subroutine can be used
               as an aid, but KWSCAN is limited to keyword recognition
               unless extra work is done outside the subroutine.

               The command-language parser, a facility for syntactic
               analysis, may be used as an aid. This consists of the
               CLPARSEGEN table generator and a table-driven procedure,
               referred to as the CLPARSER. The table generator takes a
               file containing a BNF-like grammar describing the syntax
               of the data, and produces tables in object-module form
               which are then used by the user's compiled program and
               the CLPARSER to process text.

               The parser assists in developing good command grammars
               and also in parsing other types of data records where
               efficiency is not as important as ease of coding.
               Although the parser is sufficiently quick for parsing
               command languages, it is not designed for parsing pro-
               gramming languages.   The parser is callable from any
               language generating standard S-type calling sequences
               (e.g., FORTRAN) as well as those using the MTS coding
               conventions (e.g., PLUS).

               The parser was developed at the University of British
               Columbia by Alan Ballard and is based on the BNF-Action
               parser used by the Spires project at Stanford University.
               A complete description of the parser is available in the
               file *CLPARSER.W, which will produce about 100 pages when
               copied to a printer. The parser is used by the Computing
               Center staff regularly and is very reliable.

January 1987

<u>*CMDMACLIB</u>

Contents: The default MTS command macro library of general utility
                macros.

Program Key: *EXEC

Logical I/O Units Referenced:
                None.

Description: If MTS command macros are enabled by the MTS command $SET
                MACRO=ON, this macro library is automatically attached to
                obtain a set of general utility macros.

                For documentation on using MTS command macro libraries,
                see MTS Volume 21, <u>MTS Command Extensions and Macros</u>.

January 1987

*CMS

Contents: The IBM VM/CMS operating system execution simulator.

Purpose: To allow software written for CMS to run unmodified under
                MTS.

Use: *CMS is invoked by the $RUN command or by concatenating
                the file to an appropriate driver program.

Program Key: *CMS

Return Codes: See CCMemo 448.

Description: *CMS is an extended version of *VSS.   *CMS supports a
                subset of the CMS SVC 202 and 203 functions, VM/SP
                diagnose instruction functions, virtual device support
                via the System 370 I/O machine instructions (e.g., SIO),
                and direct addressing of the VM/SP virtual page-zero
                storage locations commonly known as the NUCON area.

                The complete description of VSS is given in Computing
                Center Memo 448, "VSS: OS/VS Simulator."

| *CMSFILETYPEMAP

Contents: A standard CMS-to-MTS file-type mapping definition.

Program Key: *EXEC

Description: Since the CMS operating system allows file names with a
                 total of 16 significant characters and MTS allows only 12
                 characters in file names, it is necessary to define a
                 one-to-one mapping between the longer CMS file names and
                 the shorter MTS file names. The file *CMSFILETYPEMAP is
                 provided to define a standard mapping for MTS users who
                 run programs under the *CMS simulator or who import files
                 to MTS from a CMS system via the CMS TAPE DUMP program
                 (see descriptions of *CMSTAPELOAD and *CMSTAPESCAN in
                 this volume).

                 CMS file identifiers (fileIDs) within a give userID are
                 actually composed of a 1- to 8-character "filename" and a
                 1- to 8-character "filetype", whereas MTS fileIDs are
                 defined as a single 12-character string. By convention,
                 CMS fileIDs are therefore mapped to MTS fileIDs without
                 altering the "filename" portion and by mapping the
                 "filetype" part to a 3-character code that is appended to
                 the "filename" with a delimiting period. For example,
                 using *CMSFILETYPEMAP as the mapping file,

                     MYPROGRM FORTRAN

                 becomes

                     MYPROGRM.FOR

                 Users may concatentate their own private file-type maps
                 to *CMSFILETYPEMAP or duplicate and edit the public file
                 as needed.

                 The file is processed free-form and each line should have
                 the following elements separated by at least one blank:

                     filetype code comment

                 where "filetype" is the 1- to 8-character CMS filetype to
                 be mapped, "code" is a 1- to 3-character abbreviation for
                 the filetype, "comment" is an optional comment of any
                 length.

114.2 *CMSFILETYPEMAP

January 1987

<div align="center">

*CMSTAPELOAD, *CMSTAPESCAN

</div>

Contents: The CMS tape restore and scanning programs.

Use: These programs are invoked by the $RUN command.

Program Key: *CMSTAPELOAD and *CMSTAPESCAN

Logical I/O Units Referenced:

             For *CMSTAPELOAD:

             0   - The pseudodevice name of a magnetic tape written
                     by the CMS TAPE utility.
             SPRINT - the listing of restored files.
             SERCOM - error comments and prompts.
             GUSER - responses to prompting messages.

             For *CMSTAPESCAN:

             0   - the pseudodevice name of the CMS TAPE DUMP tape
                     to be scanned.
             SPRINT - the listing of the files on the scanned tape.

Return Codes: For *CMSTAPELOAD:

             0 - Successful return.
             8 - Error return.

             For *CMSTAPESCAN:

             Always zero.

Description: *CMSTAPELOAD restores files saved with the "TAPE DUMP"
             command available on IBM's CMS (VM/370 and VM/SP) operat-
             ing system to MTS line files.

             *CMSTAPESCAN lists the contents of a CMS magnetic tape in
             an intelligble format. A descriptive header is written
             every 50 output lines.

             Both of these programs are described in further detail in
             MTS Volume 19, Tapes and Floppy Disks.

January 1987

COBOLVS

Contents: The IBM COBOL VS compiler.

Purpose: To compile COBOL/VS source programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *COBOLVS

Logical I/O Units Referenced:
                SCARDS - COBOL source program to be compiled.
                SPRINT - source-program listing, diagnostics, etc.
                SERCOM - error messages when the compiler is invoked from
                         a conversational task or when the TERM parameter
                         is specified.
                SPUNCH - resultant object module if the DECK parameter is
                         specified. A new source deck if the FDECK or
                         CDECK parameter is specified.
                0   - resultant object module if the LOAD parameter is
                         not overridden (defaults to -LOAD).
                1   - information needed by the compiled program for
                         symbolic debugging when the SYMDMP parameter is
                         used (defaults to -SYMLOAD).

Return Codes: 0 - Successful return.
                8 - Error return.

Description: *COBOLVS contains a version of the IBM COBOL OS/VS
                compiler (release 2.3) that has been modified for use in
                MTS. Any COBOL program that will successfully compile in
                an IBM VS1 (SVS), VS2 (MVS), or VM370/CMS operating
                system should compile equally well in MTS. The object
                modules as generated by the compiler will not execute in
                the normal MTS environment because the modules contain
                SVC's that are specific to IBM operating systems. An SVC
                is an machine instruction that requests a specific
                service from the operating system.

                In order to run these object modules, the user must
                invoke the interface program *COBOLVSRUN that simulates
                an IBM VS2 operating system. This program is described
                in Computing Center Memo 439, "IBM VS COBOL under MTS."

                The following parameters may be specified in the PAR
                field of the $RUN command. Minumum abbreviations are
                underlined. The complete description of these parameters
                is given in Memo 439.

Parameter	Default
ADV or NOADV	ADV
APOST or QUOTE	APOST
BATCH or NOBATCH	NOBATCH
CLIST or NOCLIST	CLIST
COUNT or NOCOUNT	NOCOUNT
DECK or NODECK	NODECK
DMAP or NODMAP	DMAP
DYNAM or NODYNAM	NODYNAM
FDECK or NOFDECK	NOFDECK
FLAGW or FLAGE	FLAGW
FLOW[=nn] or NOFLOW	NOFLOW
LANGLVL(1) or LANGLVL(2)	LANGLVL(2)
LCOL1 or LCOL2	LCOL2
LINECNT=nn	LINECNT=57
LOAD or NOLOAD	LOAD
LSTONLY, LSTCOMP, or NOLST	NOLST
LVL=A/B/C/D or NOLVL	NOLVL
L120 or L132	L132
NAME or NONAME	NONAME
NUM or NONUM	NONUM
OPTMIZE or NOOPTMIZE	NOOPTIMIZE
PMAP or NOPMAP	NOPMAP
RESIDENT or NORESIDENT	NORESIDENT
SEQ or NOSEQ	NOSEQ
SIZE=nnnnnn	SIZE=141312
SOURCE or NOSOURCE	SOURCE
SPACE1, SPACE2, or SPACE3	SPACE1
STATE or NOSTATE	NOSTATE
SUPMAP or NOSUPMAP	SUPMAP
SXREF or NOSXREF	NOSXREF
SYMDMP or NOSYMDMP	NOSYMDMP
SYNTAX, CSYNTAX, or NOSYNTAX	NOSYNTAX
SYST or SYSx	SYST
TERM or NOTERM	TERM (terminal), NOTERM (batch)
TRUNC or NOTRUNC	NOTRUNC
VBSUM or NOVBSUM	NOVBSUM
VBREF or NOVBREF	NOVBREF
VERB or NOVERB	VERB
XREF or NOXREF	XREF
ZWB or NOZWB	ZWB

The *COBLINKEDIT program may be run to invoke some of the
more advanced features of VS COBOL, such as the dynamic
call option. This program is also described in Memo 439.

<u>*COLLATE</u>

Contents: A record collating program.

Purpose: To select records from two input files that either match
                or do not match according to user specified keys, and to
                write out one or more combinations of these records or to
                produce output records that have data combined from
                matched records.

Use: *COLLATE is invoked by the $RUN command.

Program Key: *COLLATE

Logical I/O Units Referenced:
                0   - input file containing "master" records.
                1   - input file containing "detail" records.
                SPRINT - input/output statistics.
                SERCOM - warning, error, and prompt messages.
                GUSER - error prompt responses.

Parameters: The following parameter must be included in the PAR field
                of the $RUN command:

                    <u>KEY</u>=({key-spec|(key-spec)[,(key-spec)]...})

                        where "key-spec" is:

                          begin,length[,detail_begin[,detail_length[,pad]]]

                        and

                          begin-end or detail_begin-detail_end

                        may be substituted, respectively, for

                          begin,length or detail_begin,detail_length

                        This specifies the keys for matching the records.
                        "begin" is the beginning position of the key in the
                        "master" record. "length" is the length in bytes,
                        or characters, of the key in the "master" record.
                        "end" is the ending position of the key in the
                        "master" record. "detail_begin" is the beginning
                        position of the key in the "detail" record. If
                        "detail_begin" is not specified, the value of
                        "begin" is used. "detail_length" is the length in
                        bytes, or characters, of the key in the "detail"
                        record. "detail_end" is the ending position of the
                        key in the "detail" record. If neither "detail_-

length" nor "detail_end" is specified, the length
of the key in the "master" record is used for the
length of the key in the "detail" record. "pad" is
a character to use to pad keys when the "master"
record and "detail" record key lengths are not
equal. If "pad" is a blank, comma, or right
parenthesis, it must be enclosed in single quotes
(') or double quotes ("). Otherwise, it may be a
single character or a single character enclosed in
quotes. If "pad" is not specified, a blank is
used. The input records must be ordered so that if
the keys are treated as characters or unsigned
binary numbers, they will be in ascending sequence.
Where more than one key field is specified, the key
is evaluated as though the fields were concatenated
from left to right in the order given.   Depending
on the MULTIPLEMASTERS and IGNOREMULTIPLEMASTERS
parameters, there may be only one "master" record
for each key value, or there may be more than one
for each key value. There may be any number of
"detail" records for each key value.

At least one of the following parameters must be included
in the PAR field of the $RUN command:

{<u>MATCHEDMA</u>STERS|MM}={FDname|@MD|@UM|@UD|@C|@IMM}
{<u>MATCHEDDE</u>TAILS|MD}={FDname|@MM|@UM|@UD|@C|@IMM}
{<u>UNMATCHEDMA</u>STERS|UM}={FDname|@MM|@MD|@UD|@C|@IMM}
{<u>UNMATCHEDDE</u>TAILS|UD}={FDname|@MM|@MD|@UM|@C|@IMM}

These designate the output files. "FDname" is the
name of an output file or device. "@MM" specifies
that the same output file as the one for matched
"master" records is to be used.   "@MD" specifies
that the same output file as the one for matched
"detail" records is to be used.   "@UM" specifies
that the same output file as the one for unmatched
"master" records is to be used.   "@UD" specifies
that the same output file as the one for unmatched
"detail" records is to be used.   "@C" specifies
that the same output file as the one for combined
matched records is to be used.   "@IMM" specifies
that the same output file as the one for ignored
multiple master records is to be used.

<u>C</u>OMBINED=({FDname|@MM|@MD|@UM|@UD|@IMM},
   {<u>M</u>ASTER=(combine-spec)|<u>D</u>ETAIL=(combine-spec)},...)

where "combine-spec" is:

   begin,length,outloc[,begin,length,outloc]...

118.2 *COLLATE

and

  begin-end

may be substituted for

  begin,length

This specifies the data from a matched "master" record and a matched "detail" record that is to be combined into an output record. "FDname" is the name of an output file or device. "@MM" specifies that the same output file as the one for matched "master" records is to be used. "@MD" specifies that the same output file as the one for matched "detail" records is to be used. "@UM" specifies that the same output file as the one for unmatched "master" records is to be used. "@UD" specifies that the same output file as the one for unmatched "detail" records is to be used. "@IMM" specifies that the same output file as the one for ignored multiple master records is to be used. "begin" is the beginning position of data in the "master" or "detail" record to be placed in the combined record. "length" is the length of the data to be placed in the combined record. "end" is the ending position of the data to be placed in the combined record. "outloc" is the beginning position in the combined record for the data.

{IGNOREMULTIPLEMASTERS|IMM}
    [={FDname|@MM|@MD|@UM|@UD|@C}]

This specifies that any number of "master" records with the same key value can occur. The first "master" record having each key value will match all "detail" records having the same key value. All additional "master" records having the same key value will not be matched with any "detail" re-cords. "FDname" is the name of an output file or device for the additional "master" records with the same key value. "@MM" specifies that the same output file as the one for matched "master" records is to be used. "@MD" specifies that the same output file as the one for matched "detail" records is to be used. "@UM" specifies that the same output file as the one for unmatched "master" records is to be used. "@UD" specifies that the same output file as the one for unmatched "detail" records is to be used. "@C" specifies that the same output file as the one for combined matched records is to be used. If this is the only parameter used from this group, a file or device

name must be specified. This parameter may not be
specified if the MULTIPLEMASTERS parameter   is
specified.

The following parameters may be included in the PAR field
of the $RUN command:

[NO|¬|˜]MULTIPLEMASTERS

If no multiple masters is specified, there can be
only one "master" record with each key value. Each
"master" record will match all detail records
having the same key value. If multiple masters is
specified, any number of "master" records with the
same key value can occur. Each "master" record can
match only one "detail" record. If there are more
"master" records than "detail" records or more
"detail" records than "master" records for a given
key value, the excess records are treated as
unmatched records.   The default is NOMULTIPLEMAS-
TERS. This parameter may not be specified if the
IGNOREMULTIPLEMASTERS parameter is specified.

{MASTERLENGTH|ML}=maxlen

This specifies the maximum length of the "master"
records. "maxlen" is the maximum number of bytes,
or characters, that occur in a "master" record. If
this parameter is not specified, the maximum input
length for the first file attached to logical unit
0 is used.

{DETAILLENGTH|DL}=maxlen

This specifies the maximum length of the "detail"
records. "maxlen" is the maximum number of bytes,
or characters, that occur in a "detail" record. If
this parameter is not specified, the maximum input
length for the first file attached to logical unit
1 is used.

Return Codes: 0 - Successful return.
               4 - Error detected.

Example: $RUN *COLLATE 0=MASTERFILE 1=DETAILFILE PAR=K=(1,4)
                  C=(COMBINEDFILE,M=(1,15,1, 16-26,27, 28-44,68),
                  D=(5-15,16, 16,29,38))

This reads records from MASTERFILE and DETAILFILE
and compares columns 1 through 4 of both the
"master" and "detail" records. If the records match
(columns 1 through 4 are equal), a combined record

118.4 *COLLATE

is written to the file COMBINEDFILE. The format of
the output records is:

    col 1-15   master columns 1-15
    col 16-26   detail columns 5-15
    col 27-37   master columns 16-26
    col 38-66   detail columns 16-44
    col 68-84   master columns 28-44

Since no multiple masters is in effect, no two
"master" records will have the same key in columns 1
through 4; but each "master" record may match zero
or more "detail" records. One output record will be
written for each "detail" record that matches a
"master" record.

$RUN *COLLATE 0=MASTERFILE 1=DETAIL1+DETAIL2
   PAR=K=(2-5,1) MM=MATCHED MD=@MM ML=32

This reads records from MASTERFILE and DETAIL1
concatenated with DETAIL2 and compares columns 2
through 5 of the "master" records with columns 1
through 4 of the "detail" records. "Master" records
that match "detail" records are written to the file
MATCHED. "Detail" records that match "master" re-
cords are written to the same file as the matched
"master" records, i.e., the file MATCHED.   The
maximum record length for the records in MASTERFILE
is 32 bytes, or characters.   The maximum record
length for the "detail" records is the maximum input
record length returned by the GDINFO subroutine for
the file DETAIL1.

$RUN *COLLATE 0=MASTERFILE 1=DETAILFILE PAR=K=((1,4,17),
   (5-12)) MD=MATCHEDDET UD=UNMATCHEDDET MU

This reads records from MASTERFILE and DETAILFILE
and compares columns 1 through 4 of the "master"
records with columns 17 through 20 of the "detail"
records and columns 5 through 12 of both the
"master" and "detail" records.   If the records
match, the detail records are written to the file
MATCHEDDET. If the records do not match, the detail
records are written to the file UNMATCHEDDET. Since
multiple master records is specified, there may be
more than one "master" record with the same key and
each "detail" record must match a separate "master"
record. If there are "detail" records that would
match a "master" record, but the "master" record has
already matched a previous "detail" record and there
are no more "master" records with the same key, the
"detail" record will be considered unmatched and
written to the file UNMATCHEDDET.

                                          *COLLATE 118.5

```
$RUN *COLLATE 0=MASTERFILE 1=DETAILFILE PAR=K=(1,4)
   C=(REPLACEDFILE,M=(6,10,1),D=(5,76,11))
```

This reads records from MASTERFILE and DETAILFILE
and compares columns 1 through 4 of both the
"master" and "detail" records. If the records
match, the detail record is written to REPLACEDFILE
with columns 6 through 15 of the "master" record
replacing the original first four columns of the
"detail" record.

January 1987

*COMBINE

Contents: The record-combination program.

Use: The program is invoked by the $RUN command.

Program Key: *COMBINE

Logical I/O Units Referenced:
                 SCARDS - input commands to the COMBINE program.
                 SERCOM - error comments and diagnostic messages.
                 GUSER - input commands to the COMBINE program (after
                          attention interrupt).

Return Codes: Always zero.

Description: *COMBINE is a record-combination program used to combine
                 records or parts of records from one or more input files
                 or devices, and to write the combined records onto one
                 output file or device, or to execute the combined records
                 as MTS commands.

                 Input commands to COMBINE (instructions for combining
                 records) are entered from SCARDS, one command per line,
                 e.g.,

                       $RUN *COMBINE SCARDS=command-input

                 Alternatively, a single input command may be entered via
                 the PAR field of the $RUN command, e.g.,

                       $RUN *COMBINE PAR=single-command

                 COMBINE may be used as a subroutine by concatenating
                 *COMBINE to the file containing the calling program. For
                 example, if the file GOODSTUFF contains the calling
                 program, the MTS command

                       $RUN GOODSTUFF+*COMBINE

                 will load COMBINE for use.   COMBINE is invoked as a
                 subroutine by a call of the form:

                       CALL COMBIN

                 or

                       CALL COMBIN(command)

where "command" is a halfword array with the command
length in the first halfword and with the command
starting in the second halfword. COMBINE returns to the
calling routine after executing the single command. If
no parameters are given in the call, COMBINE will read
and execute multiple commands from SCARDS. When a STOP
command is read, COMBINE will return to the calling
routine.

Commands: The following commands may be used with the COMBINE
               program.

               COPY parameter ...

                    The COPY command specifies what is to be copied and
                    where to put it. The command has the form

                         COPY FDname<C,C,...> FDname<C,C,...>... FDname

                    where "FDname" is a file or device name (line-number
                    ranges, concatenations, and modifiers are allowed).
                    The rightmost FDname is the output file or device
                    name unless only one FDname is given, in which case
                    SPUNCH is used for output.

                    Movement of columns from an input record to the
                    output record is controlled by the items listed
                    between the angle brackets, "<" and ">".   The
                    available items are listed below, with the under-
                    lined letters giving the minimum acceptable
                    abbreviations.

                    n   "n" is a column number from which to move;
                              e.g., 54
                    n-m   "n" and "m" are column numbers giving a
                              column range to move; e.g., 7-16
                    SPACE n   blanks "n" column spaces in the output
                              record; e.g., SPACE 10
                    ISPACE n skips "n" column spaces in the input
                              record; e.g., ISPACE 8
                    TAB n   tabs to the nth column in the output
                              record; e.g., TAB 33
                    ITAB n   tabs to the nth column in the input
                              record; e.g., ITAB 50
                    MOVE n   moves the next "n" columns to the output
                              record; e.g., MOVE 8
                    'xxxx'   a literal string; e.g., 'FRAXINUS'
                    OUTPUT n writes "n" output records; e.g., OUTPUT 3
                    INPUT n   reads "n" additional records from the
                              input source; e.g., INPUT 5

The spaces within each item specification are op-
tional; e.g., INPUT 1 and INPUT1 have the same
effect.

The following example illustrates the use of these
items (this command should be on one line).

```
COPY A<5,'SCILLA'> B<7,SPACE6,1-13,OUTPUT1,
  50-END,TAB15,'IRIS'> C D<OUTPUT1> Z
```

This command uses files A, B, C, and D for input and
file Z for output. The records written to Z will be
constructed as follows:

(1) A record is read from file A and column 5 is
     moved to column 1 of the Z output buffer.
(2) The characters "SCILLA" are moved to columns 2
     to 7 of the Z output buffer.
(3) A record is read from file B, and column 7
     moved to column 8 of the Z buffer.
(4) Columns 9 to 14 of the Z buffer are skipped
     (filled with blanks).
(5) Columns 1 to 13 (including 7) of B are moved to
     columns 15 to 27 of the Z buffer.
(6) OUTPUT1 causes these 27 columns to be written
     to file Z.
(7) Columns 50 to the end of the record are moved
     from B to columns 1 to ? of the Z buffer.
(8) TAB15 causes the characters "IRIS" to be over-
     layed on columns 15 to 18 of the Z buffer. (If
     column ? is short of 15, then blanks are
     inserted up to column 15.)
(9) Next, a record is read from file C and is moved
     in its entirety to columns 19 to ? of the Z
     buffer.
(10) A record is read from file D and immediately
     thereafter the current Z buffer is written.
(11) Finally, the entire record just read from D is
     moved to columns 1 to ? of the Z buffer and
     this record is written.

This entire process is repeated until an end-of-file
is received from one of the input files.

Notes:

(1) Column ranges must be ascending.
(2) The first column of an input record is always
     1.
(3) The last column of an input record may be
     designated END, LAST, or *L.
(4) If no column number or column range is given
     with an input FDname, the program assumes the

range 1-END as the last item between the angle
brackets (or the only item if no angle brackets
were used).

(5) Items listed between pairs of pointed brackets
are separated by blanks and/or commas.

(6) Groups of items may be enclosed by parentheses,
(...), with an integer repetition count preced-
ing the left parenthesis.   This causes the
enclosed items to be rescanned when the corre-
sponding right parenthesis is reached. Paren-
theses may be nested up to 10 deep.

(7) Literals may be used without the surrounding
apostrophes.   Such a literal is terminated by
any of the characters

        <>,'-0123456789()

or by any of the words INPUT, OUTPUT, SPACE,
TAB, ISPACE, ITAB, END, LAST, or *L, or their
abbreviations. Blanks may be embedded within
these literals, and leading and trailing blanks
are included in the literal.

(8) If an input end-of-file is received while the
output buffer still has characters in it, these
characters will not be written.

(9) @I modifiers are ignored on input FDnames but
are recognized on the output FDname.   The
record number of the last input record is used
as the output record number.

COMMANDS parameter

The COMMANDS command is written like a COPY command
with "COPY" replaced by   "COMMANDS".   COMMANDS
executes the output lines as MTS commands instead of
copying them to an output file or device (in fact,
there should be no output file or device included in
a COMMANDS command).

Note: An attention interrupt issued during the
processing of an MTS command terminates that MTS
command; not knowing of the attention, COMBINE then
continues with the next line of the file. Thus, it
is virtually impossible to stop the COMBINE COMMANDS
command once started.

Suppose the file NAMES contains file names.   Then
the following command would permit the files "read-
only" to CCID MINA:

        COMMANDS NAMES<'$PERMIT ',1-END,' READ MINA'>

Note the necessary blanks after "$PERMIT" and before "READ MINA".

Since the COMMANDS use of COMBINE commonly involves file names, a pseudofile of file names is provided. The pseudofile has the form

        (string)

where "string" is a partially specified file name (PSFN), and is written as in the $FILESTATUS command. That is, a question mark (?) is used to designate an arbitrary string of characters within a name. File names matching the PSFN are returned, one at a time, as if they were being read from a file. For example, suppose a user has files with the following names: EPILOBIUM, EPIMIDIUM, EPISTASIS, and EPODE. Then the command

        COMMANDS (?EPI?)<$DESTROY 1-END OK>

would execute the MTS commands

        $DESTROY EPILOBIUM OK
        $DESTROY EPIMIDIUM OK
        $DESTROY EPISTASIS OK

and the command

        COMMANDS (?I?I?)<LIST 1-END'(*L)'>

would execute the MTS commands

        LIST EPILOBIUM(*L)
        LIST EPIMIDIUM(*L)
        LIST EPISTASIS(*L)

SET option ...

    The SET commands sets switches that control some aspects of the program. The options that take ON or OFF as a value may take YES or NO instead.

    CC={ON|OFF}   Default: CC=ON

        CC controls carriage control when the output device is a printer or a terminal. When CC=ON, the first character of an output record is used for carriage control, and when CC=OFF, the output records are single-spaced.

PAD={ON|OFF}   Default: PAD=ON

    PAD controls the treatment of short input
records.   If PAD=ON and columns that do not
exist are to be moved from an input record to
the output record, then blanks will be used to
extend the input record to the required length.
If PAD=OFF, then no characters are moved to the
output record from beyond the end of a short
input record.

WARN={ON|OFF}   Default: WARN=ON

    WARN controls the printing of warning messages
for records that are too long to fit in
internal buffer of the program. WARN=ON prints
a warning message when an output record is
truncated. WARN=OFF suppresses these messages.

BLANK={ON|OFF}   Default: BLANK=ON

    BLANK controls the replacement of zero-length
records with blanks.   The value of blank is
only relevant when NULL=ON. BLANK=ON replaces
a zero-length record by a record with one blank
(" ").   If BLANK=OFF, a zero-length record is
written.

NULL={ON|OFF}   Default: NULL=ON

    NULL controls the output of zero-length (null)
records.   NULL=ON causes these records to be
written. NULL=OFF suppresses the output of
these records. For example, the command

        COPY A<1-5,OUTPUT2,6-10> B

writes columns 1 to 5 of file A to a record of
file B. Then if NULL=ON (and BLANK=OFF), a
zero-length record is written, and this is
followed by a record containing columns 6 to 10
of file A. The only effect here is that the
line numbers in file B will not be consecutive
integers, since a zero-length record deletes
the line with the same line number (if there is
one) and does not enter the file. If NULL=OFF,
zero-length records will not be written to the
output file, and the line numbers in B will be
consecutive.

BEGIN=char   Default: BEGIN=<
END=char   Default: END=>

     BEGIN and END control the column selection characters. It is necessary to change these from the defaults only when using files whose names contain "<" or ">".

EXPLAIN command

     The EXPLAIN command prints a brief explanation of a command. For example,

       EXPLAIN COPY

prints an explanation of the COPY command.

HELP

     The HELP command prints a list of commands.

CONTINUE

     The CONTINUE command may be used after an attention interrupt to continue execution of an interrupted COPY or COMMANDS command.

MTS

     The MTS command returns control to MTS. The program can be restarted with $RESTART provided that none of the following MTS commands is used in the interim: $RUN, $LOAD, $UNLOAD, $DEBUG, or $RERUN.

MCMD MTS-command

     The MCMD command executes one MTS command and returns to the program, provided the command is not one of the following: $RUN, $LOAD, $UNLOAD, $DEBUG, or $RERUN. For example:

       MCMD LIST EPIMIDIUM

In addition, any line beginning with a "$" sign is executed as an MTS command with control returning to the program. For example:

       $LIST EPIMIDIUM

STOP

     The STOP command terminates the program.

Examples: The following example executes a COPY command to combine
              columns 4 to 9 from file ABE with columns 8 to 11 of file
              PDP and writes the combined records into file HAM.

```
$RUN *COMBINE PAR=COPY ABE<4-9> PDP<8-11> HAM
```

      Contents of ABE   Contents of PDP   Contents of HAM

        1234567890   abcdefghijkl   456789hijk
        2345678901   bcdefghijklm   567890ijkl
        3456789012   cdefghijklmn   678901jklm

The following example empties the file FG, and then fills
it with columns 1 to 5 and 6 to 10 of file F and columns
7 to 9 of file G. Then pairs of records from file H are
combined and written to file HH.

```
$RUN *COMBINE
MCMD $EMPTY FG
COPY F<1-5, SPACE 3, 6-10> G<TAB 6, 7-9> FG
COPY H<1-*L, INPUT 1, 1-*L> HH
STOP
```

      Contents of F   Contents of G   Contents of FG

       abcdefghijkl   1234567890   abcde789fghij
       bcdefghijklm   2345678901   bcdef890ghijk
       cdefghijklmn   3456789012   cdefg901hijkl

      Contents of H   Contents of HH

       12345   1234567890
       67890   abcdefghijklmnopqrst
       abcdefgh
       ijklmnopqrst

The following is part of a FORTRAN program that uses
COMBINE as a subroutine to execute the command COPY
A<1-5,10-15> B.

```
INTEGER*2 CMD(11)
   ...
CALL MOVEC(19, 'COPY A<1-5,10-15> B', CMD(2))
CMD(1) = 19
CALL COMBIN(CMD)
   ...
```

      Contents of A   Resulting Contents of B

       abcdefghijklmnopqrst   abcdeklmno
       bcdefghijklmnopqrstu   bcdeflmnop
       cdefghijklmnopqrstuv   cdefgmnopq

January 1987

*COMPARE

Contents: A file comparison program.

Purpose: To compare two files.

Use: The program is invoked with the $RUN command.

Program Key: *COMPARE

Logical I/O Units Referenced:
                 SPRINT - printed output.
                 SPUNCH - control command output for *CDUPDATE or *UPDATE.
                 0   - original file or device.
                 1   - current file or device.
                 GUSER - prompting responses
                 SERCOM - prompting messages for the FDnames and error
                          comments.

Parameters: The parameters that are preceded by [NO] can also be
                 specified as "option={ON|OFF}" or as "{NO|N|-|˜}option".

                 SYNCH

                     If SYNCH is specified, a comparison between two
                     files is performed; this is the opposite of DOWNDATE
                     (see below). An attempt is made to synchronize the
                     lines of the two files. The differences are printed
                     on SPRINT, unless SPUNCH is specified in which case
                     DOWNDATE overrides the SYNCH specification. SYNCH
                     is the default.

                 WIDTH=nn

                     If WIDTH is specified, the width of the SYNCH output
                     is set to "nn", where "nn" may be from 65 to 133,
                     inclusive. By default, the width of SYNCH output is
                     set to the SPRINT maximum output length.

                 PRLEN=nn

                     If SYNCH is specified, only the first "nn" charac-
                     ters of each record are printed on SPRINT. The
                     default for PRLEN is the value MAXLEN.

                 [NO]HEXADECIMAL

                     If HEXADECIMAL is enabled, the lines of the two
                     files will be printed in hexadecimal. The default
                     is HEXADECIMAL=OFF.

ONELINE

> If ONELINE specified, records from each unit will be
> truncated so that they are printed on one line.
> This would be the same as if PRLEN is set to the
> WIDTH divided by 2 less 14. The default is
> ONELINE=OFF.

MAXLEN=nn

> For all comparisons, only the first "nn" characters
> of a record from both units are compared. The
> default is MAXLEN=32767. The user should set this
> to 72 for assembly sources.

[NO]LENCHK

> If LENCHK is enabled, lines are equal if their
> contents and lengths are equal. The default is
> NOLENCHK.

[NO]LNRCHK

> If LNRCHK is on, lines are equal if their contents
> and line numbers are equal. The default is
> NOLNRCHK.

CONTEXT=nn

> Only the first "nn" and last "nn" lines are fully
> printed. This can be zero, causing the COMPARE
> program to produce just the message indicating how
> many equal lines there are. The default is CONTEXT=
> 1; i.e., only the first and last lines are printed.

FULL

> If FULL is specified, all equal lines are printed on
> SPRINT. The default is FULL=OFF.

[NO]STAT

> Statistics, such as the number of lines in each
> file, the number of blocks, VM storage used, and CPU
> time, are printed at the end of the program. The
> default is STAT=ON.

RETRIES=nn

> If there remain false differences (sequences of
> lines that incorrectly appear different but are the
> same and cannot be resolved by the regular compari-
> son algorithm), the program will retry to resolve

them by an alternative method. The retry count specifies the number of times the program will use the alternative method. If the retry count is exhausted before resolution, the number of remaining false differences is printed in the statistics. The default is RETRIES=10.

[NO]OPT

Optimization is performed after the comparison algorithm, described in the CACM article, is accomplished. This will produce fewer blocks. The default is OPT=ON.

BLKMIN=nn

If a block would contain less than "nn" records from either unit, it is deblocked. The default is BLKMIN=5. This parameter is applicable only if OPT=ON.

BLK0MIN=nn

If a block would contain less than "nn" records from unit 0, it is deblocked. The default is 5. This parameter is effective only if OPT=ON.

BLK1MIN=nn

If a block would contain less than "nn" records from unit 1, it is deblocked. The default is 5. This parameter is effective only if OPT=ON.

BLKEQ=nn

If a block contains less than "nn" equal records, it is deblocked. The default is 2. This parameter is effective only if OPT=ON.

PCEQ=nn

If the percentage of equal records in a block is less than "nn" %, the block is deblocked. The default is 25.000%. This parameter is effective only if OPT=ON.

IC={ON|OFF|DEFAULT}

The IC parameter specifies how $CONTINUE WITH lines are to be processed. If IC=ON, $CONTINUE WITH lines are interpreted as implicit concatenation. If IC= OFF, such lines are treated as data lines. If IC=DEFAULT, the interpretation of such lines is

dependent of the @IC modifier specification on the
FDnames for logical I/O units 0 and 1. IC=DEFAULT
is the default.

The COMPARE program can be used to produce *CDUPDATE or
*UPDATE control commands. The following parameters are
applicable in producing such control-commands on SPUNCH.

DOWNDATE={CDUPDATE|UPDATE}

If the DOWNDATE=CDUPDATE is specified, *CDUPDATE
control commands will be produced on SPUNCH. If
DOWNDATE=UPDATE is specified, *UPDATE control com-
mands will be produced on SPUNCH with SEQID and
MAXLEN=72 enforced. The DOWNDATE parameter will
override the SYNCH parameter. If SPUNCH is speci-
fied, DOWNDATE=CDUPDATE is assumed if only DOWNDATE
is specified.

[NO]CDLIST

If DOWNDATE=CDUPDATE is specified, the last record
from the original version plus its line number is
recorded in a comment field for /COPY and /SKIP.
The default is NOCDLIST. This is chiefly useful for
those desiring to change *CDUPDATE commands into
those with pattern matching.

[NO]CDLNR

If DOWNDATE=CDUPDATE is specified, the line numbers
of both original and current versions are recorded
in a comment field for /BUFFER, /COPY, and /SKIP.
The default is NOCDLNR.

[NO]SEQID

Instead of COUNT=nn or patterns, LEN(72) "seqid" is
produced for /COPY and /SKIP. This is useful for
the users desiring to use the sequence IDs in
similar fashion as *UPDATE program. The default is
NOSEQID since the sequence IDs in columns 73-80 are
not likely to be unique.

COPYMIN=nn

If DOWNDATE=CDUPDATE and OPT=ON are specified, then
if the /COPY count is less than "nn", it is replaced
with /SKIP and insertions. The default is 2.

January 1987

NAME=nn

"nn" is used as a name for the *CDUPDATE /BEGIN
command. If NAME is not specified and DOWNDATE=
CDUPDATE is specified, the program will prompt the
user for the name.

Return Codes: 0 - Files are equal.
               4 - Files are not equal.
               8 - Error return.

Description: The COMPARE program produces on SPRINT the comparison
               between two files much like the SYNCH output of *APC. If
               either logical I/O unit 0 or 1 is not specified, the
               program will prompt the user. The COMPARE program is
               less expensive than *APC, but each of the files being
               compared should have less than 65,534 lines. The
               algorithm is completely different than the one used by
               current comparison programs *DOWNDATE, *SIDEDATE, *UNE-
               DIT, and *APC. It is based on "A Technique for Isolating
               Differences Between Files," Communications of the ACM,
               Vol. 21, No. 4, pp. 264-8. The COMPARE program will even
               recognize a block, a series of similar lines from both
               files. Such a block may even be found at the start of
               the first file and at the end of the second.

Example: The example on the following page illustrates the output
               from the COMPARE program.

```
#$RUN *COMPARE
#EXECUTION BEGINS
 Where is the old source?    -MAC
 Where is the current source?    -ASMTMAC
  Unit 0: -MAC                         Unit 1: -ASMTMAC
 ----------------------------------------------------------------
 =    1               MACRO        1               MACRO       =
 ====================    1 equal line  =======================
 =    3               GBLA  &#N      3               GBLA  &#N =
      4               LCLC  &S*      4               LCLC  &S,
             255,&LBL                        &LBL
 =    5       &LBL    SETC  '&L      5       &LBL    SETC  '&L =
 =           ABE'                            ABE'              =
 ====================    7 equal lines =======================
 =   13       &S      SETC  '#&     13       &S      SETC  '#& =
 =           #N.A'            NA             #N.A'            NA =
 =           ME OF IT                        ME OF IT          =
     14       .WA     AIF   ('&     14       .WA     AIF   ('&
             ARG1'(1,1) EQ '(')              ARG1'(1,1).'&ARG1'
             .REG                            (1,1) EQ '((').REG
 =   15       &LBL    MVC   &S+     15       &LBL    MVC   &S+ =
 =           4(4),&ARG1      MO             4(4),&ARG1      MO =
 =           VE INTEGER                      VE INTEGER        =
 ====================   11 equal lines =======================
 =   27       .END    MEND         27       .END    MEND      =
 Old file "-MAC" has 27 lines,
 and current file "-ASMTMAC" has 27 lines.
 Both files have 2 mismatches.
 Total virtual storage used = 3 pages.
 CPU time = 0.055343 seconds.
#EXECUTION TERMINATED   16:29:11  T=.059       $.02
```

January 1987

*COMPUCALENDAR

Contents: A calendar of computing events on the University of
              Michigan campus.

Use: The $COPY command should be used, e.g.,

                  $COPY *COMPUCALENDAR

January 1987

*CONSULTINGHOURS

Contents: A list of the hours during which consultants are avail-
                able at the Michigan Union (UNYN) and the North Universi-
                ty Batch Station (NUBS).

Use: The $COPY command should be used, e.g.,

                    $COPY *CONSULTINGHOURS

*CONSULTINGSCHED

Contents: A chart giving the hours and names of the each of the
                consultants on duty at the Michigan Union (UNYN) and the
                North University Batch Station (NUBS).

Use: The $COPY command should be used, e.g.,

                    $COPY *CONSULTINGSCHED

| ## *CRCBOOKLIST

Contents: A list of reference publications available at the Comput-
               ing Resource Center.

Use: The $COPY command should be used, e.g.,

                   $COPY *CRCBOOKLIST

Description: The Computing Resource Center Resource Room houses a
               collection of printed materials relating to microcompu-
               ters and computing at The University of Michigan.
               Materials are available to the entire University communi-
               ty. Materials may not be checked out except on a
               short-term basis for copying or for use in the ISS/CC
               labs.

*CRCSOFTWARE

Contents: A list of software available for evaluation at the
               Computing Resource Center.

Use: The $COPY command should be used, e.g.,

                   $COPY *CRCSOFTWARE

Description: The Computing Resource Center maintains a collection of
               software packages available for trial and evaluation.
               These programs for both IBM PC-compatible and Macintosh
               microcomputers can be checked out for use on specially
               designated computers in the Evaluation Area at any time
               during CRC business hours: 9:00 am to 5:00 pm Monday
               through Friday.

January 1987

*CSMP

Contents: A comprehensive control program for the CSMP (Continuous
               System Modeling Program) simulation language. See also
               the *CSMPTRAN description in this volume.

Purpose: To provide both the model stacking capability and full
               CSMP monitoring across all three CSMP phases (CSMP
               translation, FORTRAN compilation, and CSMP model
               execution).

Use: The program is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:
               SCARDS - CSMP source program if unit 1 is not assigned
                        (defaults to *SOURCE*).
               SPRINT - DECK option output if unit 2 is not assigned and
                        CSMP listings if unit 6 is not assigned
                        (defaults to *SINK*).
               1 - CSMP source program (defaults to SCARDS).
               2 - DECK option output (defaults to SPRINT).
               5 - data storage between CSMP translation and execution
                    phases (defaults to the sequential file -CSMP#5).
               6 - CSMP translation listing output, FORTRAN compilation
                    output, and CSMP execution listing output (defaults
                    to SPRINT).
               7 - FORTRAN source produced by CSMP translation phase
                    (defaults to the sequential file -CSMP#7).
               15 - PREPAR option output.

Return Codes: 0 - Successful return.
               >0 - Error return.

Description: *CSMP provides full comprehensive control over all phases
               for one or more stacked models. The format of input
               source, the use of logical I/O units, and the defaults
               are as described above and in the *CSMPTRAN public file
               description. The overall processing is as described in
               the *CSMPTRAN description, except that *CSMP maintains
               overall control and does not use *CSMPTRAN and/or *CSMPE-
               XEC. The STACK option (as described in the IBM manuals
               listed in the *CSMPTRAN description) is available in
               *CSMP and is indicated by placing the symbol STACK in
               columns 9-13 of the ENDJOB control statement. (The last,
               or only, model must terminate with an ENDJOB control
               statement which is blank in columns 9-13.)

Logical I/O units 2 and 15 <u>must</u> be specified if the CSMP source issues the DECK or PREPAR options, respectively; otherwise, these units need not be specified. Logical I/O units 5, 7, and 15 must have read-write-rewind capability, whether specified by the user or defaulted. Units 5 and 7 are initially emptied (if attached to files) or rewound (if not attached to files), and they are left in a rewound state at the exit from *CSMP. At exit, unit 5 contains the data file for the last (or only) model translated, and unit 7 contains the FORTRAN source for the last (or only) model translated. Unit 15 contains the PREPAR option output for the last model executed which specified PREPAR as an output. These units are all used, for each source-supplied model, as described in *CSMPTRAN.

During execution, *CSMP creates three temporary sequential files named -####13, -####14, and -LOAD. Files -####13 and -####14 are for internal use only and are emptied at entry and exit to *CSMP. The file -LOAD, at exit, contains the FORTRAN-compiled object modules for the last (or only) model compiled; -LOAD is emptied before each compilation. If user-specified, these three files must have read-write-rewind attributes.

During execution, *CSMP links in sequence to the CSMP translation phase, to FORTRAN (using *FORTRANG), and to the CSMP execution phase. For this linking to occur correctly, it is imperative that the MTS SYMTAB option be ON; this is the current default setting for the SYMTAB option.

Example: $RUN *CSMP 2=-T 15=PLOTFILE
                  .
               (CSMP model 1 source)
                  .
             ENDJOB STACK
                  .
               (CSMP model 2 source)
                  .
             ENDJOB STACK
                  .
               (CSMP model n source)
                  .
             ENDJOB

In the above example, the source program is read from *SOURCE* (the default for SCARDS) and the DECK option output is written to the file -T. PREPAR output is written to the file PLOTFILE.

January 1987

*CSMPTRAN, *CSMPEXEC, *CSMPLIB

Contents: The modules for the IBM CSMP (Continuous System Modeling
               Program) simulation language.   *CSMPTRAN contains the
               MTS-supplied CSMP translation phase control program.
               *CSMPEXEC contains the MTS-supplied CSMP execution phase
               control program. *CSMPLIB contains the IBM-supplied CSMP
               library modules in a subroutine library format.   (Note:
               See the *CSMP public file description for a comprehensive
               CSMP control program.)

Purpose: To translate (*CSMPTRAN) a CSMP source program into
               appropriate data and FORTRAN source files, and to execute
               (*CSMPEXEC) the various object modules composing the
               model specified by the CSMP source program.

Use: *CSMPTRAN and *CSMPEXEC are invoked by the appropriate
               $RUN commands; *CSMPLIB is used as a library file.

Program Key: *EXEC

Logical I/O Units Referenced:

               (a) by *CSMPTRAN
                   SCARDS - CSMP source deck if unit 1 is not assigned
                           (defaults to *SOURCE*).
                   SPRINT - DECK option output if unit 2 is not assigned
                           (defaults to *SINK*).
                   1 - CSMP source deck (defaults to SCARDS).
                   2 - output for the DECK option (defaults to SPRINT).
                   5 - data output for later use (defaults to -CSMP#5).
                   6 - CSMP listing and error comments (defaults to
                       SPRINT).
                   7 - FORTRAN source output (defaults to -CSMP#7).

               (b) by *CSMPEXEC
                   SPRINT - CSMP execution output if unit 6 is not
                           assigned (defaults to *SINK*).
                   5 - data input (defaults to -CSMP#5, if file
                       exists).
                   6 - CSMP execution output (defaults to SPRINT).
                   15 - PREPAR option output.

Return Codes: Always zero.

Description: CSMP is a simulation language especially designed for
               those simulation models which are expressed as differen-
               tial equations or as equivalent block diagrams. The
               language and its use are described in the following IBM
               manuals:

(1) <u>System/360 Continuous System Modeling Program:
     Application Description</u>, form H20-0240.
(2) <u>System/360 Continuous System Modeling Program
     User's Manual</u>, form H20-0367.
(3) <u>System/360 Continuous System Modeling Program
     System Manual</u>, form Y20-0111.

The user of CSMP should know the FORTRAN language and the
use of the FORTRAN compiler (*FTN or *FORTRANH) in MTS.

The input deck for CSMP is described in the afore-
mentioned IBM manuals and will be reviewed quickly here.
Briefly, the CSMP source deck format is as follows:

(1) The first deck segment contains the model de-
     scription together with the necessary control
     information for running the model; thus, this
     segment contains:
     (a) all structural statements needed to define
          the model, together with the statements need-
          ed to define the INITIAL, DYNAMIC, and/or
          TERMINAL model segments;
     (b) the control statements (translation, execu-
          tion, and output) necessary to specify the
          first simulation run;
     (c) the necessary data statements for value
          assignment.
(2) An END statement (to restart the model) or a
     CONTINUE statement (to continue the model execu-
     tion with new values).
(3) As many END and/or CONTINUE segments as desired;
     each is composed of:
     (a) data statements to give new values to various
          symbols;
     (b) additional execution and/or output control
          statements, if desired;
     (c) an END or CONTINUE statement.
(4) The last END-CONTINUE segment must be followed by
     a STOP statement; user supplied FORTRAN source
     programs may be placed after the STOP statement,
     if desired.
(5) The source deck is terminated by the ENDJOB
     statement; the STACK option, as specified in the
     IBM manuals, is <u>not</u> available and is ignored, if
     specified. (See the *CSMP public file
     description.)

The IBM manuals provide further deck details and card
format descriptions. The source deck is identical to
that described in the manuals except for the STACK option
which is unavailable in *CSMPEXEC - *CSMPTRAN usage.

Before discussing the details of the CSMP processor, we present here an overview of CSMP processing. The CSMP processor functions in three phases: translation phase, compile phase, and execution phase. During the translation phase, the CSMP deck is read and translated into temporary files:

(1) a data-control file, which contains information extracted from the data and control statements, and

(2) a FORTRAN source file, which contains a FORTRAN source program constructed from the model structural statements; this file also contains any user-supplied FORTRAN source modules (supplied optionally after a STOP statement).

In the compile phase, the FORTRAN compiler is invoked to compile the FORTRAN source file produced during the translation phase; the resulting object module is then used in the execution phase. Finally, the model is simulated during the execution phase and the specified results are produced as output. The data-control file produced during translation is used as input to this phase to provide values of variables and to specify the execution-time options and control sequence. The object module produced during the compile phase, together with various CSMP library modules, form the execution object module. The first two phases of CSMP have been separated in MTS as follows:

    *CSMPTRAN performs the translation phase
    *CSMPEXEC controls the execution phase

The compile phase is represented by a call on the FORTRAN compiler, which must be done by the user. (Another module *CSMP provides a comprehensive control program for those who desire it.)

The CSMP translation phase is initiated by the MTS command

    $RUN *CSMPTRAN PAR=Q

The CSMP source deck is read from logical I/O unit 1 (defaults to SCARDS), and the translation phase output (including error comments) is produced on logical I/O unit 6 (defaults to SPRINT). If the DECK option is specified in the source deck, then the pertinent punch output is produced on logical I/O unit 2 (defaults to SPRINT). If PAR=Q (optional) is specified, then translation errors will cause job termination at the end of the translation phase; otherwise, errors are documented, but job processing continues. The data-control file produced

during the translation phase is written on logical I/O unit 5, while the FORTRAN source file is written on logical I/O unit 7. The user may specify these logical I/O units on the $RUN command; if no specifications are given, then the defaults are:

```
5       -CSMP#5
7       -CSMP#7
```

If logical I/O unit 5 is not specified by the user, then a temporary sequential file named -CSMP#5 is created, emptied, and used. The same procedure is followed for logical I/O unit 7 and -CSMP#7. The file and/or device specified for unit 5 and/or unit 7 must have read-write-rewind capability and is handled as follows:

   (1) The unit is rewound initially (if a device) or emptied initially (if a file);
   (2) The unit is left in rewind position at the end of the translation phase processing.

If the source deck contained only data and control statements, and no structural statements, then no output is placed on unit 7; it is, however, rewound (emptied) as described above. A similar procedure is followed for unit 5 if no data and control statements are found in the source deck; note, however, that there must be at least one each of the END-CONTINUE, STOP, and ENDJOB statements in the source deck. Any FORTRAN source modules found after the STOP statement are written onto unit 7 with no modifications.

As a result of the processing by *CSMPTRAN, the CSMP source deck is processed through the ENDJOB statement, and the data-control file and the FORTRAN source file are produced. The compile phase is initiated by calling the FORTRAN compiler to compile the FORTRAN source file produced on unit 7. If unit 7 was defaulted, then the source program is in -CSMP#7 and may be processed by the MTS command

```
    $RUN *FTN PAR=S=-CSMP#7
```

which will produce an object module in the file -LOAD. The full facilities for FORTRAN are available to the user with this command as are the facilities of MTS. Note that if the CSMP source deck contained no structural statements, then no FORTRAN source was produced and the compile phase may be skipped. Note also that user-supplied FORTRAN source modules may be used after the STOP statement as described earlier, or in the compile phase itself via explicit concatenation of files when calling FORTRAN. In either case, at the end of the

compile phase there are two files of importance: the
data-control file and the object module file.

Execution (simulation) of a CSMP model is initiated by
the MTS command

    $RUN *CSMPEXEC+OBJ+*CSMPLIB

where OBJ is the name of the file which contains the
object modules produced in the compile phase; OBJ may be
a single file name or may be an explicit or implicit
concatenation of object module files and/or user library
files. Note that previously compiled user modules may be
included in this way rather than as source modules in
earlier phases of CSMP processing. Logical I/O units
used during execution are as follows:

    5   must be specified as the data-control file
          produced during the translation phase; defaults
          to -CSMP#5. The unit is not initially rewound
          during execution.
    6   all print output produced by CSMP routines
          (defaults to SPRINT).
    15   must be specified by the user only if the
          PREPAR statement is used in the model; this
          unit must, if specified, be a sequential file
          or device. This unit is initially rewound if
          needed during execution, but is not rewound at
          the end of execution. The format of the plot
          information placed on this unit is described in
          the IBM manual Y20-0111.

In addition, any logical I/O units required by user-
supplied object modules may also be specified on the $RUN
command. For users who supply FORTRAN source or object
modules with their model, it should be noted that CSMP
runs with the usual FORTRAN I/O support package. There-
fore, the normal defaults and assignments are effective
for logical I/O units. Logical I/O units 13 and 14 are
used internally by CSMP during execution, however, and
should not be used by users.

It should be apparent that it is frequently unnecessary
to proceed through all three CSMP phases in order to run
a model. One may, for example, save the two files
produced by the translation phase in permanent files.
Subsequent model changes may then be made by modifying
the FORTRAN source, recompiling, and executing with the
previously produced data-control file. Conversely, it is
possible to run a model which is already in object module
form with new data by producing the new data-control file
and using it with the old object modules. This does
require, however, that one use the symbol table part of

the previous data-control file. The IBM manuals discuss
these possibilities further; the user needs more knowl-
edge about the intermediate files and system processing
than has been presented here to use these shortcuts.

If the user wishes to supply his own control routine for
CSMP execution, he may do so by supplying a subroutine
whose name is MAINEX. This differs from the technique
described in Y20-0111, in that <u>no</u> MAIN card is needed and
that the control program is supplied as a subprogram
named MAINEX rather than as a main program. Otherwise,
the specifications for the proper functioning of the main
control program are as given in Y20-0111.

If the user desires to supply his own central integration
routine for CSMP execution, he proceeds as described in
Y20-0111. The source deck must contain the METHOD
statement specifying CENTRL as the integration method,
and the user must supply his central integration routine
as a user-supplied subprogram whose name is CENTRL.   The
manual Y20-0111 describes this process and shows an
example CENTRL routine.

As mentioned above, CSMP uses logical I/O units 13 and 14
internally, and these units should not be referred to by
users.   Actually, temporary sequential files named
-####13 and -####14 are created and used by CSMP by
attaching them to FORTRAN DSRN's 13 and 14. These files
are emptied at entry and exit to *CSMPEXEC and *CSMPTRAN,
and DSRN's 13 and 14 are, in fact logically disconnected
from logical I/O units 13 and 14.

Example: $RUN *CSMPTRAN PAR=Q
                        .
                   (CSMP source program)
                        .
                $RUN *FTN PAR=S=-CSMP#7
                $RUN *CSMPEXEC+-LOAD+*CSMPLIB

                In the above example, the source program for the
                CSMP translation phase is read from *SOURCE* (the
                default for SCARDS). The FORTRAN source output is
                written to the file -CSMP#7 which is then compiled
                by *FTN and written to the file -LOAD. The resul-
                tant program is then executed by the CSMP execution
                phase.

<u>*C87</u>

Contents: A compiler for the C programming language. This compiler
               replaces the *CBELL compiler.

Purpose: To compile C programs.

Program Key: *C87

Logical I/O Units Referenced:
               SCARDS - input of the source program.
               SPRINT - list output and object listing.
               SPUNCH - output of the object module.
               SERCOM - error messages from compiler.
               2   - additional include libraries containing header
                        files. The standard include library *C87INCLUDE
                        is always used.

Parameters: These options can be specified either in the PAR field of
               the $RUN command or in a #pragma command in the source
               program.   The PAR field and pragma option names may be
               upper-, lower-, or mixed-case.   Most options can be
               negated by prefixing them with "-", "˜", or "NO".

               When options are specified in the PAR field, their scope
               is the entire compilation unit.   When specified in a
               pragma, the scope is indicated with each description. It
               is quite likely that the scope of the pragmas will be
               changed to something more sensible in future releases.

               Of the multitude of options, only a few are frequently
               used. These are:

                   (1) SYM to aid in debugging.
                   (2) PROPER to have the compiler diagnose as many
                        suspect or nonstandard constructions as possible.
                   (3) UNIX to use when porting nonstandard programs
                        from Unix.

               The options and pragmas are:

               ASCII   Default: NOASCII

                   The ASCII option causes all character strings and
                   character constants to be converted to ASCII inter-
                   nally.   This option should be used with care.
                   Serious problems may occur if a PAR=ASCII routine
                   calls a routine that is not ASCII.

For the most part, the *C87 run-time system assumes
EBCDIC. For example, GETC, SCANF, etc., all return
EBCDIC characters, and the PRINTF and SCANF formats
must be expressed in EBCDIC. This may change at
some point in the future.

Until the library supports ASCII characters (e.g.,
for PRINTF formats), this option is of little
utility.

Pragma scope: takes effect immediately.

CHECKIOVER   Default: NOCHECKIOVER

The CHECKIOVER option causes the compiler to gener-
ate code to check that the results of all signed
integer computations fit into 32 bits.

Pragma scope: the last value is used for the entire
compilation unit.

DEFINE(x)   Default: None

This is processed as if a "#DEFINE x 1" had been
issued before the source program had started.   "x"
may be any legal identifier. "x" is not translated
to uppercase before use.

This option cannot be negated.

This option is not allowed as a pragma; use #DEFINE.

DEFINE(x=e)   Default: None

This is processed as if a "#DEFINE x e" had been
issued before the source program had started. "x"
may be any legal identifier. "e" can be any
sequence of tokens. "x" and "e" are not translated
to uppercase before use.

This option cannot be negated.

This option is not allowed as a pragma; use #DEFINE.

FILL=c   Default: NOFILL

The FILL option specifies a char-sized value that is
used at run-time to initialize each stack frame.
This option can be used in debugging when a problem
with an uninitialized variable is suspected.

The value of "c" must fit into a single character, but can be expressed as any integer constant. For example, each of the following has the same effect.

```
#PRAGMA FILL='A'
#PRAGMA FILL=0X81
#PRAGMA FILL=129
```

Pragma scope: the last value is used for the entire compilation unit.

HIDE   Default: NOHIDE

The HIDE option produces a warning if an identifier in an inner scope hides the same identifier defined in an outer scope.

I=ccid   Default: None

The I option affects #INCLUDE searches. It adds the ID "ccid" to the list of IDs to be searched when a quoted #INCLUDE file is specified. For example,

```
$RUN *C87 SCARDS=... PAR=I=W123 I=W456
```

where the source file contains the following include:

```
#INCLUDE "INK.H"
```

*C87 will first look for the file W123:INK.H. If that does not exist, it will look for the file W456:INK.H and if it is not there, it looks for the file INK.H on the current ID. As many of these options as desired may be specified and they are order dependent; the first one specified is the first one searched.

This option is useful when compiling a program developed on one ID from another ID.

#INCLUDE directives that use angle brackets are unaffected by this option.

This option cannot be negated.

Pragma scope: takes effect immediately.

LIST   Default: NOLIST
LIST=n   Default: LIST=0

    The LIST option controls echoing of source lines to
    SPRINT. The LIST identifier by itself is the same
    as LIST=1. NOLIST is the same as LIST=0.

    If LIST=1, everything except #INCLUDE files are
    listed. If LIST=2, everything including #INCLUDE
    files is listed.

    Pragma scope: takes effect immediately.

LOADNAME(cname,lname)

    The LOADNAME option causes the external name "cname"
    in the source program to be mapped into the name
    "lname" in the object module. This option can be
    used to:

    (1) map names too long for the MTS loader into
         shorter names.
    (2) allow C programs to access system names which
         do not conform to C syntax.

    Pragma scope: the same scope as "cname".

OBJECT   Default: OBJECT

    Object-code generation may be suppressed with
    NOOBJECT.

    Pragma scope: the last value is used for the entire
    compilation unit.

OBJLIST   Default: NOOBJLIST

    If OBJLIST is specified, an object listing is
    written to SPRINT.

    Pragma scope: the last value is used for the entire
    compilation unit.

OPT   Default: OPT

    When NOOPT is specified, no analysis of the interme-
    diate code is made to determine code size or
    register usage, thus code is generated using worst-
    case assumptions. NOOPT saves only a very small
    amount of compilation time.

    Pragma scope: the last value is used for the entire
    compilation unit.

PORT   Default: NOPORT

If PORT is specified, *C87 prints warnings that
indicate which parts of a program might not be
portable. This will not detect all violations;
neither does the presence of a warning guarantee a
non-portable program nor does the absence of a
warning guarantee a portable program.

The following cause a portability warning. Other
features may also be added to cause portability
warnings.

(1) Use of the FORTRAN, _RETCODE, _PRVBASE, or
    _PSEUDOREGISTER extensions.
(2) Use of the RCALL, LOADNAME, RETCODE pragmas.
(3) Casts between pointers and integers.
(4) Casts between pointers to different types of
    structures.

Pragma scope: takes effect immediately.

PROPER   Default: None

The PROPER pragma stands for a collection of other
pragmas that would all be used if one were trying to
write a maximally portable and correct program. The
pragmas set by PROPER are:

    #PRAGMA PORT STANDARD WARN TYPE PROTO

Pragma scope: takes effect immediately.

PROTO=n   Default: NOPROTO

The PROTO option controls whether prototypes are
required and how they are used. It may have the
following values:

    0 All prototypes are ignored.

    1 Permits the last parameter to be omitted
       without generating an error or warning. This
       is to be used with the ZEROARG option for
       compatibility with certain Unix implementa-
       tions (e.g., that of Sun Microsystems) that
       allow final parameters to be omitted; it also
       supplies an extra zero-valued parameter.

    2 Prototypes are used if provided, but the
       compiler does not insist on having them.

        3 All function declarations, but not defini-
          tions, are required to have prototypes.

        4 All function declarations and definitions are
          required to have prototypes.

Pragma scope: takes effect immediately.

RCALL(fname,inregs,outreg)   Default: None

The RCALL option is used only to provide an inter-
face to certain MTS functions.

This option specifies that the function "fname" is
to use R-call linkage.   The "inregs" parameter
specifies which registers are to be loaded with
parameters, and the "outreg" parameter specifies
which register, if any, contains the result.

"inregs" is specified as a sequence of hexadecimal
digits, one for each of the general registers to be
loaded with a parameter.   Each of the registers
specified in "inregs" is restricted to the range
0-7.

Similarly, "outreg" specifies either general regis-
ter R0 or R1.

For example,

    #PRAGMA RCALL(XYZ,043,1)

would declare the function XYZ to use R-call link-
age, loading the first parameter into R0, the second
into R4, and the third into R3. The result will be
returned in R1.

The register specifications are optional.

If "inregs" is omitted, parameters will be passed
with an S-type linkage, but it is still possible to
specify "outreg" as 1.

If "outregs" is omitted, the value is return in R0.

For example,

    #PRAGMA RCALL(XXX,043) /* value in R0 */
    #PRAGMA RCALL(YYY,,1) /* no parameter regs */

Pragma scope: the same scope as "fname".

RENT   Default: NORENT

The RENT option forces *C87 to check for possible
reentrancy violations. If you do not know what
reentrancy is, you do not need to use this option.

Pragma scope: takes effect immediately.

RETCODE   Default: NORETCODE

Do not use this option for new programs. The
RETCODE option is intended for users converting
*CBELL programs to *C87. To get a return code in
*C87, use the _RETCODE option on a FORTRAN call.
Future versions of *C87 may not support this option.

When given, each function in the compilation unit
will contain a predefined integer variable called
RETCODE.   Whenever a FORTRAN routine is called, the
MTS return code is stored into this variable. If no
FORTRAN routines have been called, RETCODE will
contain garbage.   RETCODE may be examined or modi-
fied by the function as much as desired.

Pragma scope: must occur before the first function
definition.

{STANDARD|STANDARD+|UNIX|UNIX+}   Default: STANDARD

These four options are especially useful in porting
programs to or from Unix systems. The two effects
have to (1) define compile-time symbols that control
which parts of the standard header files are pro-
cessed and (2) set other pragma options.

STANDARD defines only those identifiers that are in
the standard header files and defined in the Draft C
Standard.   No additional Unix symbols are defined.
This also sets the TYPE and NOUNIXCPP options.   STD
and ANSI are allowed as equivalent abbreviations.

STANDARD+ gives all of the standard identifiers from
the standard header files and additionally defines
all identifiers that would normally be defined in
the Unix version of the header file. In the few
cases that the standard and Unix definitions con-
flict, the standard definitions take precedence.
This also sets the TYPE and NOUNIXCPP options. STD+
and ANSI+ are allowed as equivalent abbreviations.

UNIX gives only those identifiers in the standard
header files that are defined in Unix. No addition-

*C87 144.7

al Draft C Standard identifiers are defined. This
also sets the NOTYPE and UNIXCPP options.

UNIX+ defines those identifiers in the standard
header files that are defined in Unix and addition-
ally defines all nonconflicting Draft C Standard
identifiers.   This also sets the NOTYPE and UNIXCPP
options.

Pragma scope: takes effect immediately.

SUMMARY   Default: SUMMARY

The SUMMARY option provides a summary of the object
code produced during the compilation of each func-
tion. It specifies the number of bytes required for
the generated code, constants, and stack.   It also
gives information about the number of unused general
and floating-point registers.

Pragma scope: the last value is used for the entire
compilation unit.

SYM   Default: SYM

If SYM is specified, SDS SYM records are generated
in the object module. See the section on debugging
*C87 programs with SDS.

Pragma scope: the last value is used for the entire
compilation unit.

TEST

Synonym for SYM.

TYPE   Default: TYPE

When TYPE is specified, *C87 will issue a warning
whenever a type is missing (defaults to INT).
Specifying NOTYPE suppresses these warnings.

Pragma scope: takes effect immediately.

UNDEF(x)   Default: None

This is processed as if a "#UNDEF x" had been issued
before the source program had started. "x" may be
any legal identifier. As many DEFINE and UNDEFs can
be issued as desired. If more than one is given,
they are processed in order. "x" is not translated
to uppercase before use.

This option cannot be negated.

This option is not allowed as pragma; use #DEFINE.

UNIX

See STANDARD.

UNIXCPP   Default: NOUNIXCPP

When UNIXCPP is specified, additional text on the
end of "#ENDIF" commands will be ignored as is done
by the Unix C preprocessor.

Pragma scope: takes effect immediately.

WARN   Default: WARN

If NOWARN is specified, warning messages are <u>not</u>
printed.   However a summary of how many warnings
were suppressed <u>is</u> printed.

Pragma scope: takes effect immediately.

ZEROARG   Default: NOZEROARG

If ZEROARG is specified, an extra INT zero argument
is appended to every parameter list and PROTO=1 set.
This is provided to mimic the behavior of the Sun
Microsystems C compiler.

Pragma scope: takes effect immediately.

Return Codes: 0 - the compilation was successful (except for possible
                   warning messages).
               4 - a file required by *C87 could not be accessed
                   (exception: #INCLUDE files that cannot be accessed
                   cause RC=8).
               8 - one or more compile-time errors were detected or an
                   #INCLUDE file could not be accessed.
              12 - the compiler ran out of memory.
              16 - *C87 internal failure.

Description: *C87 is an implementation of the proposed ANSI Standard
               C.   More complete documentation is given in Computing
               Center Memo 483, "*C87."

               A library of the standard C functions as well as some
               Unix functions is in the file *C87LIB, which must be
               concatenated on all $RUN commands of C87 object programs.

Example: $RUN *C87 SCARDS=X.C SPUNCH=X.O
               $RUN X.O+*C87LIB SCARDS=DATA

January 1987

<u>*DAVE</u>

Contents: The data-flow analyzer for FORTRAN programs.

Use: The analyzer is invoked by the $RUN command.

Program Key: *DAVE

Logical I/O Units Referenced:
                 SCARDS - the FORTRAN program to be analyzed.
                 SPRINT - output from the analyzer.
                 SERCOM - error messages and program comments.
                 GUSER - responses to prompting messages.

Return Codes: Always zero.

Description: *DAVE is a software tool for gathering information about
                global data flow in FORTRAN programs, and for identifying
                the anomalous use of data in these programs. *DAVE is a
                static analysis tool, meaning that *DAVE gathers informa-
                tion about the subject program without executing it.
                *DAVE does not require modification of the subject
                program, nor does it require intervention by the user
                during execution.   Only some initial setting of parame-
                ters which control the output is required.

                For the complete details of using the *DAVE analyzer
                program and the *DAVE.GENLIB subroutine library, see

                    (1)   Computing Center Memo 394, "DAVE"

                    (2)   Computing Center Memo 402, "DAVE.GENLIB"

January 1987

<u>*DEDIT</u>

Contents: The distribution driver-file editor.

Purpose: To build and maintain a distribution driver file.

Use: The program is invoked by the $RUN command.

Program Key: *DEDIT

Logical I/O Units Referenced:
                SPRINT - normal output from the program.
                SERCOM - prompting and error messages.
                SCARDS - user command input.
                GUSER - responses to prompts.

Return Codes: Always zero.

Parameter: The PAR field of the $RUN command may be used to specify
                the name of the driver file to be edited.

Description: This program is used to build a distribution driver file
                which, when used in conjunction with the DRIVE command in
                the *FS program, will generate a distribution tape.

                Each line in a driver file (with line number 1.001 and
                above) catalogs a component (or subcomponent) to be saved
                on a distribution tape by *FS.   The components may be
                located on disk files or on *FS or non-*FS tapes. The
                *DEDIT program may be used to insert, delete, or alter
                lines in a driver file.

                Each component cataloged in the driver file has a name
                and a number. Since some components of a distribution
                may actually consist of many "pieces" (referred to as
                subcomponents), provision is made also for subcomponent
                names and numbers. For example, component 100 might be
                the program XYZ organized as follows: component 100/1
                (i.e., subcomponent 1 of component 100) might be the
                FORTRAN source for the program, 100/2 might be the
                compiled object code, and 100/3 might be the input to
                *FORMAT for the writeup describing the use of the
                program.   In this case, the component name for all three
                might be XYZ, while the subcomponent names might be
                SOURCE, OBJECT, and WRITEUP for the subcomponent numbers
                1, 2, and 3, respectively.

                Components are stored at the line numbers corresponding
                to their component numbers.   For example, component
                103/23 is stored at line 103.023.

*DEDIT will save up to 1000 lines of descriptive comments associated with each subcomponent. These comments are stored in the driver file itself (at negative line numbers) or in a separate comment file.

The descriptive comments for each component may consist of anything the user wishes to say about the component. When a driver-file listing is produced by the LIST or PLIST commands, the comments for each component will be printed immediately after the other fields such as the component number, subcomponent names, etc. *DEDIT will automatically recognize references in the descriptive comments to other component numbers, if they are given in the form " (n/m)", where "n/m" (or just "n") represents the component/subcomponent number. For example, the descriptive comments might be

        This file contains the object for MAIN (123/1),
        SUBR1 (123/3), and SUBR2 (123/5).

Through this mechanism, *DEDIT is able to change automatically the descriptive comments whenever a component/subcomponent number is changed, e.g., by rearranging the order of the subcomponents. *DEDIT will also prevent the deletion of components from the driver file, if they are referenced in the descriptive comments of some other component (unless it is also being deleted).

Each driver-file line has several other fields which may be specified using *DEDIT including: the location of the component (file name or tape-mount information), the type code (i.e., for FORTRAN source, *FORMAT input, etc.), the save control (a control character which indicates whether the component is ready to be distributed), two fields for the names of the persons responsible for the component, and a revision level which can be used in conjunction with *FS to distribute only selected components.

Logical I/O units need not be assigned when running *DEDIT. If the PAR field is not specified on the $RUN command, the program will first prompt for the name of the driver file. Then the user may enter the commands that are described below. The first time a driver file is used, the user will be prompted for the name of the comment file (which can be the same file as the driver file, if desired).

New versions of *DEDIT will recognize a driver file that was built by a previous version and automatically convert the old file to the current format, if necessary.

January 1987

Commands: The following commands are available for *DEDIT. The
                 underlined portion of each command name may be used as an
                 abbreviation.

       ASSIGN tapename [{6250|1600} mount-info]

               This command associates a string of up to 40
               characters of tape-mount information with an output
               tape name. If no mount information string is given,
               any string previously associated with the specified
               output tape name is deleted. This command is used
               in conjunction with the REDISTRIBUTE command.

       COMMENT [{ADD|REPLACE|EDIT}]

               This command adds, replaces, or edits the comments
               for the current component. If ADD or REPLACE is
               specified, comments are entered on one or more
               separate lines. A null line or an end-of-file
               should be used to indicate the end of the comments;
               the program will return to *DEDIT command mode.   If
               EDIT is specified, the MTS file editor is invoked to
               allow full editing of the comments. If no parameter
               is given, EDIT is assumed.

       COMPONENT {n[/m]|*F|FIRST|*L|LAST}

               This command makes component "n/m" the current
               component for editing. "n" is the component number;
               "m" is the optional subcomponent number. If no "/m"
               is given, /1 is assumed. Alternatively, the first
               or last subcomponent in the driver file may be
               specified.

       CPRINT

               This command prints the comments for the current
               component on SPRINT.

       DELETE [item, item, ...]

               This command deletes the specified components or
               subcomponents and their comments or the current
               component, if no items are given. Each item may
               consist of:

                       n   - an entire component
                       n/m - a specific subcomponent
                       n[/m]...n[/m] - a range of components or
                             subcomponents

DUPLICATE n

This command duplicates the current component "n" times (or once, if "n" is omitted) immediately after the current line. The remaining subcomponents in the component (if any) will be renumbered to follow the duplicated lines.

EDIT filename

This command causes *DEDIT to switch to a new driver file "filename".

FOR item, [item, ...] /command

This command permits the application of a /command to more than one component. Since a /command is used to replace the contents of a particular field (e.g., the component name) in the current component, the FOR command may be used to replace a field in a class of lines (e.g., all components assigned to specified persons, components with specified revision levels, components with the specified save controls, subcomponents of specified components). Each item may consist of:

```
n          - an entire component
n/m        - a specific subcomponent
n[/m]...n[/m]   - a range of components or
                  subcomponents
person     - a person name (up to 9 characters)
REVL="a" - restrict to revision level "a"
SAVE="b" - restrict to save control "b"
ALL        - the entire driver file
```

Note that each item must be followed by a comma. More than one item of the same type may be given to specify several components, persons, revision levels, and/or save controls.

INIT

This command regenerates several internal dictionaries used by *DEDIT. This process may take several minutes to complete for large driver files, especially if there are large numbers of component cross-references in the comments.

INSERT [n[/m]] [WITH PROMPT]

This command inserts a new component after the specified subcomponent (or the current component, if no subcomponent number is given) and makes it the

new current component. When the "WITH PROMPT"
parameter is specified, the user will be prompted to
enter the following component fields:

        Revision level code
        Save control
        Component and subcomponent names
        Component type
        Goodness classification
        File name or tape-mount information
        Person responsible for component

All other component fields will remain blank except
for the SHARE code which will be set to the code of
the local installation. If the "WITH PROMPT" param-
eter is not given, all fields in the inserted
component will be blank.

LIST [ON FDname] [item, item, ...]

This command produces a listing of components with
their associated comments on "FDname" (or SPRINT, if
omitted) for the specified person(s), components
with the specified revision level(s), components
with the specified save control(s), and/or component
numbers. If no items are given, the entire contents
of the driver file is listed. The listing is
ordered by component number. Each item may consist
of:

        n   - an entire component
        n/m   - a specific subcomponent
        n[/m]...n[/m] - a range of components or
                   subcomponents
        person - a person name (up to 9 characters)
        REVL=a - restrict to revision level "a"
        SAVE=b - restrict to save control "b"

Note that each item must be followed by a comma.
More than one item of the same type may be given to
specify several components, persons, revision
levels, and/or save controls.

Several modifiers may be given with the LIST command
to control the form of the listing. These are
described with the COMMENTS, LISTING, and REFERENCES
keywords of the SET command.

MERGE filename [[CHANGING] tapename1 [TO] tapename2]

This command will read lines from another driver
file specified by "filename" and add them to the
current driver file, optionally changing the output

tape-name fields as specified by "tapename1" and
"tapename2". As lines are added, their subcomponent
numbers will be renumbered, if necessary, since each
line will be added as a new subcomponent to the end
of the appropriate component.

MOVE item [TO] n[/m]

The subcomponent(s) specified by the "item" parame-
ter are moved to immediately follow the subcomponent
specified by "n/m". If "/m" is omitted, the subcom-
ponents are moved to the beginning of component "n".
Other subcomponents in the affected component(s)
will be renumbered as necessary so as to maintain
contiguous subcomponent numbers in each component.
The "item" parameter may consist of:

        n   - an entire component
        n/m   - a specific subcomponent
        n[/m]...n[/m] - a range of components or
                     subcomponents

MTS [command]

This command returns control to MTS command mode if
no parameter is given. *DEDIT may be reentered with
the MTS $RESTART command. If a parameter is given,
it is executed as an MTS command.

PLIST [ON FDname] [item, item, ...]

This command produces a listing of components with
their associated comments on "FDname" (or SPRINT, if
omitted) for the specified person(s), components
with the specified revision level(s), components
with the specified save control(s), and/or component
numbers. If no items are given, the entire contents
of the driver file are listed. A separate listing
is produced for each person and each listing is
printed in alphabetical order by component name.

Each item may consist of:

        n   - an entire component
        n/m   - a specific subcomponent
        n[/m]...n[/m] - a range of components or
                     subcomponents
        person   - a person name (up to 9 characters)
        REVL="a" - restrict to revision level "a"
        SAVE="b" - restrict to save control "b"
        ALL   - the entire driver file

Note that each item must be followed by a comma.

More than one item of the same type may given in order to produce a listing of several components, persons, revision levels, and/or save controls.

Several modifiers may be given with the PLIST command to control the form of the listing. These are described with the COMMENTS, LISTING, and REFERENCES keywords of the SET command.

PRINT

This command prints the current component on SPRINT.

REDISTRIBUTE

This command is used in conjunction with the ASSIGN command to indicate that the current component has not changed since the last distribution and that the current location of the component is on the last distribution tape. The file-name field of the current component is replaced by the tape-mount information (see the ASSIGN command) associated with the component's output tape-name field (6250 or 1600 bpi), followed by "FSNAME=" and the FS name field for this component. In addition, the file-number field is replaced by the output tape file-number field (6250 or 1600 bpi).

SET keyword keyword ...

The legal keywords are:

CMDSCAN={MTS|UNAMBIGUOUS|AMBIGUOUS}

The CMDSCAN keyword controls the minimum acceptable abbreviations for *DEDIT commands. It defaults to the same setting as the MTS $SET CMDSCAN keyword. If scanning is set to UNAMBIGUOUS, then the minimum-length command abbreviations are as shown by the underlining in the command descriptions. If scanning is set to AMBIGUOUS, then the following commands have the shorter abbreviations as shown:

        COMPONENT
        DELETE
        MTS
        PRINT
        STATISTICS

COMMENTS={ON|OFF}

The COMMENTS keyword controls whether the descriptive comments are printed as part of the listings produced by the LIST and PLIST commands. The default is ON, which prints the comments. The @COMMENTS and @NOCOMMENTS modifiers may be appended to the LIST and PLIST commands to override the effect of this keyword for the duration of a single command.

LISTING={FULL|PARTIAL}

The LISTING keyword controls whether the listings produced by the LIST and PLIST commands contain all of the fields from each component or only the most important fields. The default is FULL, which prints all fields. The @FULL and @PARTIAL modifiers may be appended to the LIST and PLIST commands to override the effect of this keyword for the duration of a single command.

REFERENCES={ON|OFF}

The REFERENCES keyword controls whether the listings produced by the LIST and PLIST commands include a list of all the subcomponents whose comments reference each listed component. The default is ON. The @REFERENCES and @NOREFERENCES modifiers may be appended to the LIST and PLIST commands to override the effect of this keyword for the duration of a single command.

WARNINGS={ON|OFF}

The WARNINGS keyword controls whether warning and advisory messages are to be printed. The default is ON.

STATISTICS

This command prints summary statistics about the current status of the driver file on SPRINT (see save control field description below).

STOP

This command terminates the program.

UPDATE [{ALL|/command ...}]

>   This command is similar to the /commands (described
>   later) in that it allows specified fields in the
>   current component to be replaced.   If the ALL
>   parameter (or no parameter) is given, *DEDIT will
>   print the current contents of each of the component
>   fields listed below, prompting the user for a
>   replacement for each field.   If a null line is
>   entered in response to a prompt, the corresponding
>   field will remain unchanged. Otherwise, the field
>   will be replaced with the string entered. If the
>   string is too long for the field, it will be
>   truncated and an error comment will be printed. The
>   fields that will be prompted for are:

>>   Revision-level code
>>   Save control
>>   Component and subcomponent names
>>   Component type
>>   Goodness classification
>>   File name or tape-mount information
>>   Person responsible for component

>   Alternatively, any single /command (with or without
>   the slash) may be given as a parameter on the UPDATE
>   command. In this case, *DEDIT will print only the
>   current contents of the field corresponding to that
>   /command and then prompt for a replacement string.
>   If a null line is entered, the field will remain
>   unchanged. This form of the command is useful when
>   the user knows exactly which field needs to be
>   replaced, but wants to see the current value before
>   entering a replacement. The function is similar to
>   entering the /command directly except that *DEDIT
>   will print the current value before it is replaced.

VISUAL [{n[/m]|*F|FIRST|*L|LAST} [{VIEW|DETAILED}]]

>   This command is available only on terminals for
>   which MTS full-screen support is available, e.g.,
>   OP/VIS Ontels and IBM 3270s.   It provides for
>   full-screen editing of components in two formats:

>   Detailed Format

>   In this format, all user-changeable driver file
>   fields for the specified component (or the currently
>   active component if no parameter is given) are shown
>   on the screen and may be directly changed. In
>   addition, the descriptive comments for the component
>   are shown (to the extent they will fit on the
>   screen) and program-function (PF) keys permit the

forward and backward "windowing" of the comments as well as allowing the editing of the comments using the normal MTS file editor. In addition, a PF key is provided to switch the display into view format.

The display will function in detailed format if a complete component/subcomponent number is given on the VISUAL command and the VIEW parameter is not specified at the end of the command, e.g., VISUAL 204/6. In addition, detailed format will be used to display the currently active component if no parameters are given on the VISUAL command, or if the DETAILED parameter is specified at the end of the command.

View Format

In this format, the screen is filled with abbreviated information from a sequence of subcomponents with each screen line presenting information from a different subcomponent. Only the component number, name, subcomponent name, type, save control, revision level, and part of the file name/tape-mount information field are shown. PF keys allow the forward and backward "windowing" of the part of the driver file being displayed, as well as switching into view format for a specified subcomponent. In addition, the currently active component number is shown at the top of the screen.

The display will function in view format if a component number without a subcomponent number is given on the VISUAL command, e.g., VISUAL 204, if a nonexistent subcomponent number is specified, or if the VIEW parameter is specified at the end of the command.

While in visual mode, the following program functions are provided by PF keys:

Component Back 10 Move back 10 subcomponents or until subcomponent /001 is reached. If the display is already at /001, then the last subcomponent of the previous component will be displayed.

Component Back 1 This function is the same as Component Back 10 except that the display is only moved back one subcomponent.

Component Fwd 10   Move forward 10 subcomponents or until the last subcomponent of the current component is reached. If the display is already at the last subcomponent, then the first subcomponent of the next component will be displayed.

Component Fwd 1   This function is the same as Component Fwd 10 except that the display is only moved forward one subcomponent.

Switch to Detailed Fmt   This function switches the display screen to Detailed Format for the currently active component (generally the component pointed to by the screen cursor).

Switch to View Fmt   This function switches the display screen to View Format.

Edit Comments   This function causes the MTS editor to be invoked in visual mode for the comments associated with the currently active component.

Execute command   This function causes the command in the command area at the bottom of the screen to be executed.

Comments Back Scrn   This function causes the comments for the currently active component to be "windowed" backward by an amount equal to the number of lines visible on the screen less one.

Comments Fwd Scrn   This function causes the comments for the currently active component to be "windowed" forward by an amount equal to the number of lines visible on the screen less one.

Comments Fwd Line   This function causes the com-
                         ments   for   the   currently
                         active component to be "win-
                         dowed" forward by one line.

The user may exit DEDIT visual mode at any time by
pressing either the PA1 (ATTN) or PA2 (EOF) key. If
the PA2 (EOF) key is used, the current command line
at the bottom of the screen will be printed on the
screen after visual mode is exited.

The following charts show the function assignments
of the other PF keys on the terminal.

Detailed Format

PF1/13 Component Back 10	PF2/14 Switch to View Fmt	PF3/15 Comments Back Scrn
PF4/16 Component Fwd 10	PF5/17 Edit Comments	PF6/18 Comments Fwd Scrn
PF7/19 Component Fwd 1	PF8/20 Execute Command	PF9/21 Comments Fwd Line
PF10/22 Component Back 1	PF11/23	PF12/24

View Format

PF1/13 Component Back 10	PF2/14 Switch to Detail Fmt	PF3/15
PF4/16 Component Fwd 10	PF5/17 Edit Comments	PF6/18
PF7/19 Component Fwd 1	PF8/20 Execute Command	PF9/21
PF10/22 Component Back 1	PF11/23	PF12/24

[-]n

> This command moves the current line pointer forward
> or backward "n" lines in the driver file. Each line
> corresponds to a subcomponent in the driver file.

$string

> Any command beginning with a dollar sign ($) is
> interpreted as an MTS command.

/Commands: The following commands each replace a field in the
current component or subcomponent. Each of these com-
mands must begin with a slash "/".   The form of each
command is

> /command [string]

where "string" is used to replace the specified field in
the component or subcomponent. If no string is given,
the field is replaced by blanks. Normally, the string is
assumed to begin with the first nonblank character after
the end of the command (there must be at least one
blank).   However, since this prohibits any field from
beginning with blanks or slashes, the program will
process a _string_ beginning with a slash as follows: the
first character _after_ the slash will be the first
character of the string used for the replacement.

The FOR command may be used to apply a /command to more
than one line. The UPDATE command may be used to update
all of the commonly used fields; the user will first be
shown the current contents of each field and then
prompted for a replacement.

For each command given below, the length is the maximum
number of characters that may be entered into the
associated field. If a string is too long for the field,
it will be truncated and an error comment will be
printed. Abbreviations for the commands are underlined.

Command	Length	Field
/DISKNAME	21	file name on distributed disk pack(s)
/FDNAME	59	file name or tape-mount information
/FILE#	4	file number (if tape)
/GOODNESS	1	goodness classification

/LPERSON	9	name of <u>local</u> person responsible for component
/NAME	25	component name
/OT62	6	output tape name (6250 bpi)
/OT16	6	output tape name (1600 bpi)
/OF62	4	output tape file number (6250 bpi)
/OF16	4	output tape file number (1600 bpi)
/PERSON	9	name of person responsible for component
/REVL	1	revision-level code
/SAVECNTL	1	save control
/SEQID	4	sequence-field identification
/SHARE	3	installation SHARE code
/SUBNAME	14	subcomponent name
/TYPE	3	component type

Format: The following is a description of the format of the driver file. The column-number range is given in parentheses.

Component Name (1 - 25)

   The name should not contain parentheses.

Subcomponent Name (27 - 40)

   The name should not contain parentheses.

Subcomponent Number (41 - 43)

   The number must be in the range of 1 to 99 inclusive.

Component Type (45 - 47)

   The component types (left-justified) are as follows:

```
B    - Binary (BTK=TeX-generated DVI file)
C    - MTS commands
        CM=command macros
        CML=command macro library
        CE=commands editor
```

160   *DEDIT

```
|                           D     - Data
|                                  DF=$MAKE dependency file
|                                  DQL=definition library for Plus
|                           H     - Help information
|                                  HC=CLParser format
|                                  HF=FSHelp format
|                           L     - Listing (same extensions as used for Source)
|                           LE  -   Linkage editor commands
|                           LML -   Load module library
|                           M     - Messages (program message file)
|                           O     - Object
|                                  OC=unlinkedited
|                                  OE=linkedited
|                                  OL=library
|                                  OV=MVS load modules in VSS format
|                           P     - Printed output
|                                  PL=line printer ready
|                                  PP=page printer ready
|                           S     - Source code
|                                  SA=assembler
|                                      SAL=assembler source library
|                                  SAW=ALGOLW
|                                  SB=WEB
|                                  SC=C language
|                                  SF=Fortran
|                                      SF6=*FTN (Fortran66)
|                                      SF7=VS/FORTRAN (Fortran77)
|                                  SG=GOM
|                                  SI=ICON
|                                  SJ=Algol W
|                                  SL=LISP
|                                  SN=COBOL
|                                  SP=PL/I
|                                  S3=PL360
|                                  SQ=PLUS
|                                      SQL=PLUS source library
|                                  SR=CLPARSER grammar
|                                  SR=REDUCE
|                                  S4=RATFOR
|                                  SS=Spitbol
|                                      SS4=Snobol4
|                                      S*=Snostorm
|                                  SW=Pascal
|                                  SX=XPL
|                                  SY=YACC
|                                  SZ=Prolog
|                           U     - Update deck
|                                  UC=*CDUPDATE
|                                  UE=$EDIT
|                                  UI=*IEBUPDAT
|                                  UU=*UPDATE commands
|                                  UB=WEB update
|                           W     - Writeup input
```

```
|                      WF=Format
|                      WT=Text360
|                      WLK=LaTeX
|                      WTK=TeX
|                      WX=Textform
|         X    - Xerox 9700
|                      XF=form definition
|                      XJ=job description library
|                      XT=font definition
```

   Installation Code (48 - 50)

      Codes for other MTS installations are as follows:

      HPK - Hewlett Packard CCC
      RIO - CBPF, Rio de Janeiro, Brasil
      RPI - Rensselaer Polytechnic Institute
      SFU - Simon Fraser University
      UBC - University of British Columbia
      UD - University of Durham
      UM - University of Michigan
      UNE - University of Newcastle upon Tyne
      UQV - University of Alberta
      UZ - University of Zagreb
      WSU - Wayne State University

   Goodness Classification (52)

      The goodness classifications (normally used only for
      programs) are given as follows:

      G - good
      O - OK
      S - shaky (has bugs)
      B - bad (needs rewriting)

   Component Number (53 - 56)

      The number must be in the range of 1 to 9999 inclusive.

   Save Control (58)

      The save control classifications are as follows:

        - normal (blank)
      # - not to be distributed by *FS
      @ - information incomplete
      = - temporary hold
      > - very large component (causes special processing
          by *FS)

File Name (59 - 117)

  The file name (or tape-mount information) specifying
  the location from which *FS is to obtain the component.

  This field is normally the name of the file from which
  *FS is to save the component. However, if the com-
  ponent is obtained from a tape, this field is given as

      rack [volume] [VLO] ['tape id'] [keywords]

  where "rack" is the rack number, "volume" is the volume
  name for a labeled tape, VLO is used to denote a
  volume-label-only tape, "tape id" is the external
  (paper) label, and "keywords" are any of the following:

  FSNAME=name This implies that the tape is an *FS tape.
               "name" may be given with or without an FS
               version number.

  POSN=name   This is for non-FS tapes. "name" speci-
               fies the file name (for a labeled tape).
               POSN=*n* may not be used; instead the file
               number "n" should be put into the File
               Number field (see below).

  FMT=fmt   This is used to specify the blocking
               format for non-FS tapes, e.g.,
               FB(8000,80).

File Number (118 - 121)

  The file number if the component is to be obtained from
  a tape (optional for labeled and *FS tapes).

Sequence Identification (123 - 126)

  The sequence ID to be applied to the component as it is
  saved by *FS.

Person (128 - 136)

  The name of the person responsible for the component at
  the installation specified by the SHARE code (see
  above).

Output Tape Name (6250 bpi) (138 - 143)

  This field is added by *FS and is not normally edited.

Output Tape File Number (6250 bpi) (144 - 147)

  This field is added by *FS and is not normally edited.

FS Name (149 - 180)

  The *FS name generated for the component.   This field
  is added by *FS and is not normally edited.

FS Version Number (182 - 184)

  The *FS version number of the component. This field is
  added by *FS and is not normally edited.

File Type (186 - 190)

  The file type (LINE or SEQ). This field is added by
  *FS and is not normally edited.

Maximum Record Length (192 - 196)

  The length of the longest record in the component.
  This field is added by *FS and is not normally edited.

Size (197 - 200)

  The size of the component (in pages if the device type
  is PAGE, in tracks if DISK). This field is added by
  *FS and is not normally edited.

Device Type (202 - 205)

  The device type from which the component was obtained
  (PAGE for non-FS tapes and files, DISK for items
  obtained from older (before the page-formatted file
  system) *FS tapes). This field is added by *FS and is
  not normally edited.

Date (207 - 219)

  The date on which the component was saved. This field
  is added by *FS and is not normally edited.

Time (221 - 228)

  The time at which the component was saved. This field
  is added by *FS and is not normally edited.

Comment Line Number (230 - 234)

  The line-number range for comments associated with this
  component (implicitly negative). This field is auto-
  matically maintained by *DEDIT.

Local Person (235 - 243)

  The name of the person responsible for the component at
  the local installation. This field is used to specify
  a "local contact" person for a component normally
  maintained by a person at another installation (as
  specified by the Person and SHARE code fields).

Revision Level (245)

  The revision-level code may be used to mark a subset of
  a larger driver file.   The FOR, LIST, and PLIST
  commands have provision for operating only on specified
  revision level codes, as does *FS.

Disk Name (246 - 266)

  The name of the public or private file where the
  component is located on the distributed disk pack(s).
  This field is used only for distributions of MTS, where
  tapes containing disk-pack images are distributed in
  addition to the normal *FS distribution tapes.

Output Tape Name (1600 bpi) (267 - 272)

  This field is added by *FS and is not normally edited.

Output Tape File Number (1600 bpi) (273 - 276)

  This field is added by *FS and is not normally edited.

Comment Forward Reference Index (277 - 282)

  This field is used by *DEDIT as a pointer to a list of
  component numbers referenced by the comments of this
  component.

Comment Backward Reference Index (283 - 288)

  This field is used by *DEDIT as a pointer to a list of
  component numbers referenced by the comments of this
  component.

164.2 *DEDIT

January 1987

*DITTO

Contents: The ditto-master preparation program.

Purpose: To convert output from *FORMAT, *TEXTFORM (line-printer formatted output), or *TEXT360 into a format suitable for preparing a ditto master.

Use: The program is invoked by the $RUN command.

Program Key: *DITTO

Logical I/O Units Referenced:
               SCARDS - text input.
               SPRINT - reformatted output.
               SERCOM - error diagnostics and prompting messages.
               GUSER - responses to prompting messages.

Return Codes: 0 - Successful return.
               8 - Error return.

Description: This program produces output suitable for any terminal with an operational backspace character; this includes most types of terminals except for the Teletype termi-nals. The output is printed on a page-by-page basis. Typical applications of this program are to produce ditto masters by inserting the masters into the terminal carriage and having the output typed directly onto the master, and to produce output on sheets of bond paper.

               If SPRINT is assigned to a device type other than any of the above, overprint characters will be printed on the line   below   the   characters   they are supposed to overstrike.

               When the program is invoked, it will initially print the message:

                   TYPE "RETURN" WHEN READY FOR TYPED COPY

               At this point, the ditto master or other sheet of paper should be placed in the terminal carriage, lined up one line above where the first line is to start. When this is done, the RETURN key should be depressed.   The   first page of the document will be typed; when this page is finished, the program will pause. At this point, a new master may be inserted and the entire procedure repeated. This may be continued for as many pages as desired.

The program will interpret the first character of each input line as a carriage-control character in the same manner as for line printers. Lines with carriage-control characters that would cause overprinting (e.g., for underlining) will be converted into a sequence of print characters and backspace characters which will produce the same result. Pages are identified by the "1", ";", or ":" logical carriage-control characters.

If any difficulty arises in the middle of a page, an attention interrupt may be given which will interrupt the printing of that page. The program will rewind the input file to the beginning of the current page. This page may then be repeated after inserting a new sheet.

The program will terminate after the last page in the input file is printed or when the user enters an end-of-file instead of entering RETURN for a new page to begin.

The program can also be used to scan selectively for a particular page in the document. A page scan is requested by typing an integer page number before entering RETURN. This number is used to count the pages from the beginning of the input file. Note that this may not necessarily correspond to any document page; e.g., entering 5 will cause the fifth page from the beginning to be printed whether or not it is document page number 5. When using this feature, it is advisable to first insert a sheet of scrap paper into the terminal, type the desired page number, and then RETURN. The page will begin to print and the user can verify that it is the desired page. The page may then be interrupted, the ditto master inserted, and then restarted at the beginning. After typing a page in this manner, the program will print the following page if another RETURN is entered.

January 1987

<u>*DI3000.IG</u>
<u>*DI3000.PRINTER</u>
<u>*DI3000.TX4010</u>
<u>*DI3000.TX4113</u>

Contents: A library of FORTRAN-callable subroutines for drawing and
               manipulating pictures.

Use: The appropriate DI-3000 library file should be concate-
               nated to the file containing the user's object modules on
               the $RUN command, e.g.,

                   $RUN object+*DI3000.TX4010

Program Key: *EXEC

Logical I/O Units Referenced:
               5   - Input (default: *SOURCE*)
               6   - Output (default: *SINK*)
               90   - Segment Storage Output (default: *SINK*)
               91   - Segment Storage Input (default: *SOURCE*)
               92   - Metafile System Output (default: *SINK*)
               93   - Metafile System Input (default: *SOURCE*)
               94   - Device Driver Output (default: *SINK*)
               95   - Device Driver Input (default: *SOURCE*)
               99   - Scratch

Description: DI-3000 is a library of FORTRAN-callable subroutines for
               drawing and manipulating pictures.   It was written by
               Precision Visuals, Inc., and is an implementation of the
               ACM SIGGRAPH Proposed Core Standard. Graphics programs
               written using these subroutines should be able to be run
               on any machine that has an implementation of the ACM
               SIGGRAPH Proposed Core Standard. The interface to the
               user's program is device independent, so a program that
               uses DI-3000 can be run without alteration on any of the
               supported devices.

               In MTS, DI-3000 is made up of the DI-3000 basic level,
               DI-3000 Extended, the Contouring System, and the Metafile
               System; it supports the Tektronix 4010 and 4113 termi-
               nals, a printer (line or page), and all devices supported
               by *IG.

               For a more complete description of DI-3000, see Computing
               Center Memo 463, "Using the DI-3000 Graphics Library in
               MTS."

| *DOCINFO

Contents: The searchable online documentation database.

Purpose: To allow users to search for documentation produced by
               the Computing Center staff including the U-M Computing
               News. The database will gradually be expanded to include
               documentation produced by other departments within the
               Information Technology Division.

Use: The program is invoked by the $RUN command.

Program Key: *DOCINFO

Return Codes: None.

Description: *DOCINFO provides easy access to Computing Center docu-
               mentation on a variety of topics. Users can send most
               documents to the Xerox 9790 page printers without typing
               *PAGEPR commands. Short documents such as QuickNotes can
               be read at the terminal as well as printed.

               The database will present the user with a brief introduc-
               tion, and then ask the user to select the viewing mode.
               Users unfamiliar with the PF keys for their terminals, or
               users connected to UMnet through a 1200 bps or slower
               connection, will want to select line mode.

               Users can search for documents by subject, area, and
               title.   The SUBJECT index searches a controlled list of
               descriptive terms. The AREA index searches a short list
               of general computing topic areas.   The TITLE index
               searches for any single word or combination of words in
               the title.   Before searching by subject or area, users
               should first scan the subject list or area list for the
               correct term.

               Each document in *DOCINFO is assigned one area and an
               average of three subjects. Generally, an area search
               retrieves more documents than a subject search.

               If the title of a document is known, the title index will
               perform the most efficient search. Any single word or
               combination of words in the title can be used with the
               title index.

               Whenever a database record references a printer-ready
               file, users are given the option to route the document to
               a Xerox 9790 page printer. The necessary commands are
               embedded in the program, so users do not have to know how

| to construct page printer commands. As many as twenty
| documents can be queued for printing in one session.
|
| Users are also given the option to read short documents
| on their terminal or microcomputer screens. The program
| enables users to page through a screen-readable document
| in a controlled manner, one screen of text at a time. A
| note alerts users when the end of the document is
| reached. Users can scroll forward or backward at will.
| This is the case for a variety of terminals, including
| those lacking a conversation buffer.

168.2 *DOCINFO

January 1987

<u>*DVIAPS</u>

Contents: DVIAPS converts input from a DVI file such as that
                 produced by the TeX text-processing program into a form
                 suitable for typesetting by the APS-Micro 5
                 phototypesetter.

Use: DVIAPS is invoked by the $RUN command.

Program Key: *DVIAPS

Logical I/O Units Referenced:
                 SERCOM - prompts, error recovery prompting, and
                          diagnostics.
                 GUSER - prompt response input.

                 DVIAPS generally references all files explicitly by name.
                 The use of SCARDS and SPUNCH on the $RUN command is
                 optional.

Return Codes: Always zero.

Description: A complete description of the use of *DVIAPS with the TeX
                 text-processing language is given in the following
                 publication:

                     Computing Center Memo 813, "Using Dviaps, TeX, and
                         the APS-Micro 5 Phototypesetter on MTS."

                 *DVIAPS takes as input a device-independent (DVI) file
                 produced by *TEX and converts it to a form that can be
                 sent to the APS-Micro 5 phototypesetter. *TYPEQ must be
                 used to queue the document for the phototypesetter in
                 order to produce output on coated photographic paper.

                 The following command, without any logical I/O unit
                 specifications, causes DVIAPS to prompt for the input
                 file with the prompt "DVI file>":

                     $RUN *DVIAPS

                 Logical I/O unit specifications can also take the follow-
                 ing form:

                     $RUN *DVIAPS SCARDS=inputfile SPRINT=outputfile

                 This does not assume any file-naming conventions. The
                 following form of the $RUN command may be used to specify
                 the input file on the $RUN command:

```
$RUN *DVIAPS PAR=xxxx.dvi
```

Specification of the ".dvi" extension on the file name is optional. Once run, the program will prompt for options that control the appearance of the document or the way that *DVIAPS works. However, options specified on the $RUN command are ignored.

Options: The following options may be specified in response to the "DVIAPS option>" prompt. The minimum abbreviation is one letter and they may be given in either upper- or lowercase. These options are fully described in Computing Center Memo 813.

HELP or ?

    Provide a list of all options with their appropriate syntax.

OPTIONS

    Report on the current state of all relevant options.

@filename

    Read options from the file "filename". The default is to not read a user-options file.

MAGNIFICATION number

    Override the DVI magnification to "number". The default is 1000.

TITLE text

    Add "text" to the between-page slug. The default text is "DVIAPS Textset, Inc."

WHITESPACE min-top max-top min-height

    Set white space (margin) at the top and bottom of the page. The defaults are .25 inches, .25 inches, and .0 inches.

INTERACTIVE

    This allows production of partial sets of pages.

NUMBER n

    Output is limited to the number of pages "n" at most. The default is no limit.

STARTING_APS_PAGE_NUMBER n

    Specify starting DVIAPS page number "n". The de-
fault is 1.

PAPERWIDTH width

    Specify the width of typesetter paper to be used.
The default is 7.28 inches.

XY xoffset yoffset crop-x crop-y crop-ht crop-wd

    This command is used to build up "virtual pages" and
to position crop marks. The defaults are .25 inches
for "xoffset" and "yoffset" and no crop marks.

LAYOUT n

    The layout command selects use of a page slug and/or
a "cut lines" mark. The default is to do slugs and
cutmarks.

ERASE

    This command restores all previous XY commands to
the default.

RESET

    This command resets all the initial program defaults
to their original (non-MTS) values.

CYCLE n

    This command starts the first page of DVI output at
other than the first XY location. The default is 0.

DUMP n startpos

    This command causes the next "n" APS ICL commands
emitted by DVIAPS to be displayed. The default is
0.

January 1987

<u>*DVIXER</u>

Contents: DVIXER converts input from a DVI file such as that
                 produced by the TeX text-processing program into a form
                 suitable for printing on the Xerox 9700 page printer.

Use: DVIXER is invoked by the $RUN command.

Program Key: *DVIXER

Logical I/O Units Referenced:
                 SERCOM - prompts, error recovery prompting, and
                          diagnostics.
                 GUSER - prompt response input.

                 DVIXER generally references all files explicitly by name.
                 The use of SCARDS and SPUNCH on the $RUN command is
                 optional.

Return Codes: Always zero.

Description: A complete description of the use of *DVIXER with the TeX
                 text-processing language is given in the   following
                 publication:

                     Computing Center Memo 810, "An Introduction to TeX."

                 *DVIXER takes as input a device-independent (DVI) file
                 produced by *TEX and converts it to a form that can be
                 sent to the Xerox 9700 page printer.   That output
                 includes nonprinting Xerox 9700 character codes. *PAGEPR
                 must be used to produce readable hard-copy output.

                 The following command, without any logical I/O unit
                 specifications, causes DVIXER to prompt for the input
                 file with a "**" prompt:

                     $RUN *DVIXER

                 Logical I/O unit specifications can also take the follow-
                 ing form:

                     $RUN *DVIXER SCARDS=inputfile SPRINT=outputfile

                 This does not assume any file-naming conventions.   The
                 following form of the $RUN command may be used to specify
                 additional parameters that affect the appearance of the
                 document or the way that *DVIXER works:

$RUN *DVIXER PAR=xxxx.dvi [% parameter list]

Specification of the ".dvi" extension on the file name is optional.

Parameters: The following parameters may be specified in the PAR field of the $RUN command or on the same line as the file name given in the response to the "**" prompt. The parameter list must be preceded by a %-sign, followed by the individual parameters, separated by blanks or commas. Minimum abbreviations are underlined.

DIALOG or NODIALOG

    The DIALOG parameter requests interactive prompting for the specification of the most useful other parameter values. The default is NODIALOG.

STARTCOUNTS=n (or STARTPAGES=n)

    The STARTCOUNTS parameter specifies the starting page for translation from the DVI file. The default is the first page (which may not be #1).

MAXPAGES=n

    The MAXPAGES parameter specifies a maximum number of pages to translate. All pages are translated if this parameter is omitted.

DATEMARK or NODATEMARK

    The DATEMARK parameter places a date and time mark at the top of each page. The default is NODATEMARK.

ORIGIN=(n,m)

    The ORIGIN parameter sets coordinates on the paper (in points) of the upper-left corner of text. "n" is the horizontal coordinate and "m" is the vertical coordinate. The minimum horizontal coordinate is 0 points and the minimum vertical coordinate is 20 points. The default is ORIGIN=(72,72).

ONESIDED or TWOSIDED

    The ONESIDED parameter forces onesided copy. The default is TWOSIDED.

PGFLINE=n

The PGFLINE parameter sets the default PGF line. The default is PGFLINE=50. The available choices are 50 (the default), 51, 52, and 53.

SORT={SHELL|INSERT}

The SORT parameter specifies the desired sort algorithm. The default is SHELL.

RESOLUTION=n or LOWRESOLUTION

The RESOLUTION parameter specifies the vertical movement size (in x-dots) for resolution cutoff. The default is 5000 (i.e., maximum possible resolution). For low resolution, the LOWRESOLUTION parameter may be specified (which takes no value).

FONTSUBS=FDname

The FONTSUBS parameter specifies the file containing an alternate font substitution table file name. This allows the substitution of an alternate set of fonts to approximate those used in the DVI file.

SHOWERRORS

The SHOWERRORS parameter specifies that extra error information should be displayed when an error occurs.

DUMPFRAGMENTS

The DUMPFRAGMENTS parameter displays information about each fragment as it is being processed. Normally, such information is not displayed as it is intended only for the debugging of *DVIXER.

*EBCDASC

Contents: The EBCDIC-to-ASCII translation program.

Purpose: To translate lines from EBCDIC to ASCII code.

Use: The program in invoked by the $RUN command.

Program Key: *EBCDASC

Logical I/O Units Referenced:
                 SCARDS - EBCDIC records to be translated.
                 SPUNCH - ASCII records resulting from the translation.

Parameters: At most, one of the following parameters may be specified
                 in the PAR field of the $RUN command. The default
                 parameter is TDAY88.

                 TDAY88   Characters are translated according to the TDAY88
                          translation table. This translation corresponds
                          to that used in MTS for translating the IBM Code
                          Page 37 codes used in MTS (EBCDIC) to ISO 8859/1
                          8-bit codes (ASCII).

                 EVEN   Characters are translated according to a 7-bit
                          translation table; even-parity ASCII characters
                          are produced, i.e., characters with the high-
                          order parity bit set to either one or zero such
                          that the number of one-bits in each eight-bit
                          character is even.

                 ODD   Characters are translated according to a 7-bit
                          translation table; odd-parity ASCII characters
                          are produced.

                 OFF or ZERO

                          Characters are translated according to a 7-bit
                          translation table; ASCII characters with the
                          high-order parity bit set to zero are produced.

                 ON or ONE

                          Characters are translated according to a 7-bit
                          translation table; ASCII characters with the
                          high-order parity bit set to one are produced.

Return Codes: 0 - Successful return.
               4 - Invalid parameter specified.

Description: Records are read from SCARDS, translated according to the
                appropriate table as specified by the parameter field,
                and written to SPUNCH. An end-of-file on SCARDS termi-
                nates the program.

                For EVEN, ODD, OFF, ZERO, ON, or ONE, all EBCDIC
                characters that would translate to codes in the top half
                of the ISO 8859/1 table (hex 80-FF) are translated into
                the ASCII SUB character (hex 1A).

                See the *ASCEBCD description in this volume for a program
                to translate from ASCII to EBCDIC.

                The TDAY88 translation table is given below. This table
                is also contained in the file DOC:ALLCHARTABLE.

Example: $RUN *EBCDASC SCARDS=OLDFILE SPUNCH=*T* PAR=EVEN

                In the above example, EBCDIC records are read from
                the file OLDFILE, translated to even-parity ASCII
                using the 7-bit translation table, and written to
                the tape *T*.

January 1987

*ENCRYPT

Contents: A program to scramble files.

Purpose: To provide additional security for programs and data by
               encoding them according to a "secret" key supplied by the
               user.

Use: The program is invoked by the $RUN command.

Program Key: *ENCRYPT

Logical I/O Units Referenced:
               SCARDS - the data to be encoded or decoded.
               SPUNCH - the data that has been encoded or decoded.
               SERCOM - prompting messages.
               GUSER - user responses to prompting messages.

Parameters: One of the following parameters may be specified in the
               PAR field of the $RUN command. If no parameter is given,
               the user will be prompted. Underlining indicates the
               minimum acceptable abbreviation.

               ENCRYPT   The data are to be encrypted.
               DECRYPT   The data are to be decrypted.

Return Codes: 0 - Successful return.
               4 - The encryption keys do not match, thus no encryption
                   done (in terminal mode only).
               8 - Error return.

Description: The data security is provided by the secret "key" and not
               by hiding the algorithm. For those interested, the
               source code for the program may be found in the MTS file
               *ENCRYPT(1000).

               The following should be noted about the program:

               (1) It is invoked by the MTS command:

                        $RUN *ENCRYPT SCARDS=input SPUNCH=output

               (2) The program will ask (on SERCOM) whether the user
                    is encrypting or decrypting. The reply is given
                    by typing "E" or "D", respectively (on GUSER).
                    The user is then prompted for the secret "key."
                    The key is then entered and may be of any length,
                    the longer, the better. The same key is used to
                    decrypt the data later. Internally, the typed
                    key is converted to a 64-bit key before use.

(3) All lines read and written by the program are
     with the TRIM I/O modifier disabled.   The user
     should be careful to remember to specify @-TRIM
     on the "from" and "to" files when $COPYing the
     encrypted file to some other place.   If the
     encrypted form of a line happens to end with two
     or more blanks, the original data will not be
     recovered if the blanks have been trimmed.
(4) Lines shorter than 8 bytes will be padded on the
     right with blanks to make them 8 bytes long
     before encrypting. The algorithm processes the
     data in 8-byte blocks and there is no unambiguous
     way to flag short lines. All lines 8 bytes or
     longer have the same length in the encrypted
     form.
(5) The user may wish to place some sort of sequence
     ID (using *MVC for example) in his lines before
     encryption so that identical lines of data do not
     have the same encrypted form. The user may also
     wish to pad his lines in some way to hide the
     actual length of the lines. Alternatively the
     records may be blocked first using the "copy"
     feature of the *SORT program. This would prevent
     padding of short lines and provide additional
     security due to the longer length of the blocks.
(6) To encrypt using the same line file for both
     input and output, the MTS I/O modifier @I <u>must</u> be
     specified on the file name assigned to SPUNCH
     (see the second example below).

The algorithm uses a subroutine (called CRYPT) which
takes three items as input.   The first is a 64-bit
(8-byte) word of data, the second is a 64-bit key, and
the third is a switch indicating whether the data are
being encrypted or decrypted. The subroutine performs a
loop 32 times using two different bits of the key at each
iteration. The first of these two bits indicates which
of two translate tables is used to translate (using the
machine translate instruction TR) the 8-byte word.   The
two translate tables consist of distinct random permuta-
tions of all byte values from 0 to 255. The second bit
is used to determine whether the 64 bits are to be
rotated 3 or 5 bits. Finally, the iteration number is
added to the low-order end of the 64-bit word.

For the inverse process, all of the operations are
reversed. The iteration number is subtracted, the rota-
tions are done in the opposite direction, and two
translate tables are used which are the inverses of the
original ones.

The lines in the file are first padded to 8 characters
using blanks. Starting with the first 8 bytes of the

line, they are encrypted using the above subroutine. The
next 8-byte block to be encrypted starts with the fifth
byte. Thus, the first half of this block is the
resulting last 4 bytes of the previous encryption fol-
lowed by 4 new bytes to encrypt.   This operation is
repeated moving down the line by 4 bytes at a time until
the end is reached. In case the line is not a multiple
of 4 bytes long, the last 8 bytes are encrypted. This
same procedure then starts with the right end of the line
working back toward the front. In this way, any single
bit in the line will affect the outcome of every other
bit in the line.   To assure that all lines have a
different encryption, at least one bit should be dif-
ferent in every line.   An appended sequence number
mentioned above would suffice.

Examples: $RUN *ENCRYPT SCARDS=DATA SPUNCH=FUNNYDATA

                $RUN *ENCRYPT SCARDS=X SPUNCH=X@I

January 1987

<u>*ENDJUNK</u>

Contents: This file contains an LCS loader input record referring
               to the system low-core symbol dictionary LCSYMBOL and an
               LDT load-terminate loader input record.

Use: The LCS and LDT input records may be either copied to the
               end of the user's object module

                    $COPY *ENDJUNK OBJECT(LAST+1)

               or concatenated to the object file on the $RUN command

                    $RUN OBJECT+*ENDJUNK

Program Key: *EXEC

Description: Normally, the dynamic loader will search the system
               library *LIBRARY for unresolved symbols. The file *LI-
               BRARY has an LCS loader input record at the end of it
               which causes the search of the low-core symbol dictionary
               if the symbol is not found in *LIBRARY. By appending LCS
               and LDT input records to the end of the user's object
               module, the search of the low-core symbol dictionary is
               made and the loading operation is terminated without
               making a search of *LIBRARY if there are no other
               unresolved symbols. This will reduce the loading time
               for the program. Because of the presence of the LDT
               input record, any object modules concatenated to the
               object module containing the LDT input record will not be
               loaded.

190 *ENDJUNK

January 1987

<u>*EXPL</u>

Contents: The "extended" version of the XPL compiler.

Use: The compiler is invoked by the $RUN command; the submoni-
                tor is concatenated to the object module produced by the
                compiler.

Program Key: *EXPL

Logical I/O Units Referenced:
                SCARDS - source input to the compiler.
                SPRINT - printed output from the compiler.
                SPUNCH - the object module produced by the compiler.
                7-14   - these units correspond to the XPL FILE function
                         files 1 to 8. If they are not assigned on the
                         RUN command, virtual memory space will be used
                         for them and, consequently, the space will be
                         released when execution terminates.

Return Codes: 0 - Successful return or no severe errors.
               n - "n" severe errors.
               8 - Internal EXPL error.

Parameter: The FSA=nP (or SIZE=nP) parameter may be specified in the
               PAR field of the $RUN command.   "n" is the number of
               pages the submonitor should use for the free string area.
               The default size is 15 pages. When running large
               programs, it is advisable to specify 25 pages.

Description: The primary advantage of "extended" XPL over *XPL is that
               a program may consist of several independently compiled
               procedures.   For a complete description of the exten-
               sions, see Computing Center Memo 416, "EXPL-Extended
               XPL."

               The submonitor, *EXPLIB, which supports extended XPL
               object modules, performs the same functions as the
               standard XPL submonitor, but has some extra features.
               These features are also described in Memo 416.

               The EXPL source for the XPL proto-compiler skeleton is
               contained in the file *EXSKELETON. This source is
               suitable as a basis for constructing a one-pass compiler
               employing the MSP(2,1;1,1) parsing strategy. The skele-
               ton consists of a parser and a lexical scanner for a
               small grammar language.   The parsing tables must be
               replaced with the tables produced by the *ANALYZER
               program. *EXSKELETON is basically identical to *SKELETON

except that it is divided into separately compilable
procedures for use with the *EXPL compiler.

The EXPL source for the EXPL compiler is contained in the
file *EXCOM.

Example: $RUN *EXPL SCARDS=EXPLSOU SPUNCH=EXPLOBJ
                  $RUN EXPLOBJ+*EXPLIB

In the above example, the source of the XPL program
is read from EXPLSOU and the resulting object module
is written into EXPLOBJ; EXPLOBJ and the submonitor
are then executed.

The input entered by the user is in lowercase; system output is in
uppercase.

```
#$run *filedump
#EXECUTION BEGINS

 ENTER NAME OF FILE TO DUMP:
 -file(1,2)
 LINE NO.        1    ;  80 CHARACTERS
 02C5E2C4404040404040003040400001C4E4D4D7C640404000000000400007DC
 E2C5D9C3D6D44040020000004040404E2C3C1D9C4E240400200000040404040
 4040404040404040C4D4D7C6F0F0F0F1
 LINE NO.        2    ;  80 CHARACTERS
 02C5E2C4404040404040003040400004C7C5E3C6C44040400200000040404040
 D9C5C1C44040404002000000040404040E2D7D9C9D5E340400200000040404040
 4040404040404040C4D4D7C6F0F0F0F2
 ENTER NAME OF FILE TO DUMP:
 $endfile
#EXECUTION TERMINATED
```

January 1987

<u>*FLOWCHART</u>

Contents: The MTS version of the IBM flowcharting package.

Purpose: To draw flowcharts according to a user-specified flow-
                chart program.

Use: The program is invoked by the $RUN command.

Program Key: *FLOWCHART

Logical I/O Units Referenced:
                SCARDS - flowchart source program.
                SPRINT - source listings, label file listings, cross-
                         reference file listings, flowcharts, and diag-
                         nostic messages.

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command.   The parameters must be
                separated by a comma. The underlined portion may be used
                as an abbreviation.

                <u>S</u>OURCE   Source listing is printed (the default if
                              SPRINT is not assigned to a terminal).
                <u>NOS</u>OURCE   No source listing is printed (the default if
                              SPRINT is assigned to a terminal).

                <u>L</u>ABELS   Label file listing is printed (the default
                              if SPRINT is not assigned to a terminal).
                <u>NOL</u>ABEL   No label file listing is printed (the de-
                              fault if SPRINT is assigned to a terminal).

                <u>X</u>REF   Cross-reference file listing is printed (the
                              default if SPRINT is not assigned to a
                              terminal).
                <u>NOX</u>REF   No cross-reference listing is printed (the
                              default if SPRINT   is   assigned   to   a
                              terminal).

                <u>F</u>OOTNOTES   Automatic   footnoting   is   enabled   (the
                              default).
                <u>NOF</u>OOTNOTES   Automatic footnoting is disabled.

                SEQ   Sequence checking is enabled (the flowchart
                              source program must be in card image for-
                              mat). No sequence checking is the default.

                SBS   Standard IBM Block Shapes (the default).
                ABS   Standard ANSI Block Shapes.

CHARS=x...   Special characters for drawing lines and
block shapes. The defaults are ⌐,⌐,⌐,⌐,>,V,
<,A,—,|. If any of these are to be changed,
they must all be listed in the following
order: upper-right corner, lower-right cor-
ner, upper-left corner, lower-left corner,
right arrow, down arrow, left arrow, up
arrow, horizontal line, and vertical line,
with no punctuation or intervening blanks.
If this is not done, undesirable results
will occur.

Return Codes: 0 - Successful return.
               >0 - Error return.

Description: For the complete description of this program (program
               number 360A-SE-22X), see the IBM publication <u>System/360</u>
               <u>Flowchart User's Manual</u>, form GH20-0293.

               All output from this program uses machine carriage
               control.   This output is designed for printers using the
               TN print train. Since the Computing Center's printers
               are run with a line spacing of 6 lines per inch, each
               flowchart will take up to 3 pages to print.

Example: $RUN *FLOWCHART SCARDS=DATA SPRINT=CHART PAR=NOSOURCE

               In the above example, the flowchart source program
               is read from the file DATA and the output is written
               to the file CHART. No source listing is printed.

Sample: The following is the sample input for one flowchart,
               followed by its output. This chart was printed at 8
               lines per inch so that it might be photo-reduced to fit
               on one page in this volume.

```
        HEAD1 /* S/360 FLOWCHART */;
        HEAD2 /* PHASE SIX */;
        HEAD3 /* SIDE ANALYSIS - SASHL ROUTINE */;
SASHL:  T ENTER;
%B1SA2: P (SAPLIN) COMPUTE SIDE PREF CODE AND INSERT SIDE PREFERENCE
CODES IN SASWS;
        P MOVE CODE TO SASWS + 1;
        P INITIALIZE SIDE PREFERENCE CODES AND LOAD FROM SIDE;
%E1SA2: D IS IT T, L, R, OR B;
        IF NO GOTO SASW3(XR,ET); ELSE YES;
        P CHANGE SIDE PREFERENCE CODE TO SPECIFIED CODE;
SASW3:  D SWITCH;
        IF BR GOTO A2SA2(XR,ET); ELSE NOP;
        M SET SASW3 TO BRANCH;
        P REVERSE SIDE PREFERENCE CODE FOR TO SIDE;
        P LOAD TO SIDE;
        GOTO E1SA2(XL,EL);
```

```
%A2SA2:  M SET SASW2 TO NOP;
         P MOVE SIDE PREFERENCE CODE TO SAWORK AND REVERSE SIDE;
SAACL:   D CMP RECORD BEING CHECKED TO CARD TABLE;
         IF EQ GOTO SAACDF(XR,EL); ELSE UNEQ;
         D IS TABLE RECORD TO BLOCK;
         IF YES GOTO SAACDT(XR,EL); ELSE NO;
SAALL:   D DOES T-L REC FLOWLINE GO TO REC;
         IF YES GOTO E3SA2(XR,EL); ELSE NO;
%F2SA2:  P INCREMENT TO NEXT FLOWLINE;
         D ARE WE AT END OF RECORD;
         IF NO GOTO SAALL(XL,EL); IF YES GOTO SACDLP(XR,EL);
%H2SA2:  P (SETFMS) SET FROM SIDE;
         P (SETTOS) SET TO SIDE;
         GOTO SALPR(XR,EL); SKIP C+4;
%E3SA2:  P IC SIDE SPECIFIED;
         P (SAREPL) REMOVE SIDE FROM PREF TABLE SAWORK;
         GOTO F2SA2(XL,ER);
SACDLP:  P INCREMENT TO NEXT TABLE RECORD;
         D ARE WE AT END OF TABLE;
         IF YES GOTO H2SA2(XL,ER); IF NO GOTO SAACL(XR,EL);
SALPR:   D ARE ALL FLOWLINES PROCESSED;
         IF YES GOTO K4SA2(XR,ET); ELSE NO;
         P SET POINTER TO NEXT FLOWLINE;
         GOTO B1SA2(XB,ET); SKIP 3;
SAACDT:  D IS FROM SIDE BLANK;
         IF YES GOTO G4SA2(XR,ET); ELSE NO;
         P IC R4 WITH FROM SIDE CHARACTER;
         P (SAREPL) REMOVE SIDE FROM PREF TABLE SAWORK+4;
%G4SA2:  P INCREMENT TO NEXT FLOWLINE;
         D ARE WE AT END OF RECORD;
         IF NO GOTO SAACDT(XL,EL); ELSE YES;
%J4SA2:  P RESET ADDR TO FIRST FLOWLINE IN RECORD;
         GOTO SAALL(XR,ET);
%K4SA2:  T EXIT; BREAK; SKIP 2;
SAACDF:  D IS FROM SIDE BLANK;
         IF YES GOTO F5SA2(XR,ER); ELSE NO;
         P IC R4 WITH FROM SIDE CHARACTER;
         P (SAREPL) REMOVE FROM SIDE PREF TABLE SAWORK;
%F5SA2:  P INCREMENT TO NEXT FLOWLINE;
         D ARE WE AT END OF RECORD;
         IF NO GOTO SAACDF(XL,EL);
         IF YES GOTO J4SA2(XB,ET);
         END;
```

January 1987

FORTRANVS

Contents: The IBM VS FORTRAN compiler.

Purpose: To compile FORTRAN 77 source programs.

Use: The VS FORTRAN compiler is invoked by the $RUN command.

Program Key: *FORTRANVS

Return Codes: 0 - Successful return.
                4 - Warning messages printed.
                8 - Errors detected.
                12 - Serious errors detected.
                16 - Internal compiler error.

Description: The VS FORTRAN compiler supports the most recent standard
                for FORTRAN 77 published by the American National Stan-
                dards Institute (ANSI). It also supports extensions to
                the language and contains features that are not available
                with FORTRAN 66 compilers (FORTRAN-G and FORTRAN-H).

                See the description of *FORTRANVS in the section "VS
                FORTRAN" in MTS Volume 6, FORTRAN in MTS, for further
                details on using this compiler.

January 1987

*FORUM

Contents: A computer-based conferencing program.

Purpose: To provide a mechanism for holding on-line private or
                 public meetings or conferences. FORUM allows many dis-
                 cussions within each conference and for extended commen-
                 tary on any given discussion.

Use: The program is invoked by the $RUN command.

Program Key: *FORUM

Logical I/O Units Referenced:
                 SPRINT - Printed output

Return Codes: 0 - Successful return.
                 8 - Error return.

Description: A complete description of FORUM is given in MTS Volume
                 23, Messaging and Conferencing in MTS, and Computing
                 Center Memo 447, "FORUM - A Computer Conferenceing
                 Program." The description here gives only a general
                 outline of the capabilities and features of the program.

                 The FORUM program maintains a database of on-line meet-
                 ings or conferences. Like round-table meetings each
                 FORUM meeting usually involves several people working
                 together for some specific purpose.

                 Unlike round-table conferences there is no need for the
                 people taking part to gather in the same place or at the
                 same time.   FORUM keeps a record of all the discussions
                 going on.

                 FORUM can manage any number of CONFERENCEs within each of
                 which there may be many DISCUSSIONS. Each DISCUSSION is
                 composed   of   an   INTRODUCTION and a collection of
                 RESPONSES. (Note: The capitalization in this paragraph
                 emphasises the terms used in the FORUM documentation.)

                 A conference is usually 'managed' by someone (its creator
                 or anyone given equivalent rights by the creator) who
                 enters one or more discussions of an introductory nature.
                 (FORUM can be instructed to direct new members of a
                 conference to a particular discussion which should be
                 read first).

                 A conference may be private to a small group or it may be
                 public, that is, open to any user of MTS.

Once a conference is begun, any participants may enter
further discussions or respond to any already started.
It is usually a good policy to separate clearly different
issues in different discussions.

After discussions have been entered and responses made,
it is possible for users with access to the conference to
print out discussions and/or responses that have certain
characteristics, such as all responses made to a given
person or all responses made after a given date.

Parameters: A specific conference name may be given in the PAR field
to restrict access to one named conference if the user
participates in several FORUM conferences.

January 1987

*FS

Contents: A file-save and file-restore program.

Purpose: To save files (sequential or line) on magnetic tape and
               to restore them from the tape by name with the original
               line numbers preserved.

Use: FS is invoked by the $RUN command.

Program Key: *FS

Logical I/O Units Referenced:

SCARDS - The source of commands to *FS (defaults to
         SOURCE).
SPRINT - A listing of the directory of files on the tape,
         prompting messages, and program comments
         (defaults to *SINK*).
SERCOM - Error comments and help information (defaults to
         MSINK).
GUSER - Responses to prompting messages.
0   - The pseudodevice name of the magnetic tape to be
         used by *FS. The tape must be a 9-track tape
         and, if files are to be saved, must be mounted
         with RING=IN.
1   - The tape used for output from COPY commands.
         This must be a 9-track tape mounted with
         RING=IN.
2   - Documentation for DSAVE. If not specified,
         documentation will be read from GUSER.

Parameters: The following parameters may be specified in the PAR
               field of the $RUN command.

               INIT   The INIT parameter must be specified when the
                         tape on unit 0 is to be used in conjunction
                         with *FS for the first time. The tape will be
                         rewound and two tapemarks will be written,
                         indicating the tape is empty.

               RECOVER   The RECOVER parameter may be specified when
                         INIT is omitted. With this option, if a record
                         cannot be read, FS will attempt to copy or
                         restore as much of the data as possible in the
                         record. If RECOVER is not specified, the
                         record causing the error will be skipped.
                         Using RECOVER may cause a file to be copied or
                         restored incorrectly.

             DIST   The DIST parameter specifies that FS is being
                       used in distribution mode.

             FIX   The FIX parameter may be used to delete a last
                       incomplete file on a labeled tape. This is
                       used when the program was stopped abnormally in
                       the middle of a save operation causing the last
                       file to be incomplete. This parameter has no
                       effect on an unlabeled tape.

             QUIT   The QUIT parameter causes the user to be signed
                       off after *FS terminates if any SAVE or RESTORE
                       operation resulted in an error.

Return Codes: 0 - Successful return.
               4 - Warning messages were issued.
               8 - Error messages were issued.

Commands: The following commands are available for *FS.   The
               underlined portion of each command name may be used as an
               abbreviation.

               CANCEL tape-file[(version)]
               CANCEL [(m)] [(n)]

                   The CANCEL command cancels all queued RESTORE or
                   COPY command operations for the file(s) named.

               COPY tape-file[(version)] [newname] [pdn]
                    [PAR={INIT|RESCAN},DOCUMENT,RECOVER]
               COPY [(m)] [(n)] [pdn]
                    [PAR={INIT|RESCAN},DOCUMENT,RECOVER]
               COPY (m) [newname] [pdn]
                    [PAR={INIT|RESCAN},DOCUMENT,RECOVER]

                   The COPY command copies files from the FS tape to
                   the copy tape.

               DELETE tape-file[(version)]
               DELETE [(m)] [(n)]

                   The DELETE command is the same as the CANCEL
                   command.

               DLIST [pdn] tape-file[(version)] [SORT=NAME] [ON FDname]
               DLIST [pdn] [(m)] [(n)] [SORT=NAME] [ON FDname]

                   The documentation saved on "pdn" by DSAVE for all of
                   the files named is printed.

DSAVE filename [tape-file]

    The DSAVE command is similar to (and may be used in
place of) the SAVE command except that up to 2000
characters of "documentation" may be saved with the
file.

FSTAPE pdn [PAR={INIT|RESCAN}[,FIX][,QUIT][,RECOVER]]

    The FSTAPE command reassigns the FS tape (the tape
normally attached to logical I/O unit 0).

HELP [{command|topic}]

    The HELP command provides on-line assistance.

LIST [pdn] tape-file[(version)] [SORT=NAME] [ON FDname]
LIST [pdn] [(m)] [(n)] [SORT=NAME] [ON FDname]

    The directory information for the specified files on
tape "pdn" is printed on "FDname".

MTS

    Execution of the *FS program is suspended and a
return to MTS command mode is made without proces-
sing queued FS requests.

QDISPLAY

    The QDISPLAY command displays all of the files that
are queued to be restored or copied.

QPROCESS

    All queued restore and copy requests are processed.

RESTORE tape-file[(version)] [filename]
RESTORE (m) (n) [pdn]
RESTORE (m) [filename]

    The RESTORE command restores the tape-files speci-
fied to the filenames specified.

SAVE filename [tape-file]

    The SAVE command specifies that the file "filename"
is to be saved using "tape-file" as the name of the
file on the tape.

SET option ...

The following options may be specified:

HELP={LINE|SCREEN}

This option specifies whether help information provided by the HELP command is printed in line mode or full-screen mode. The default is full-screen mode for terminals that support it.

TERSE={ON|OFF}

This option specifies whether messages from *FS are given in full or abbreviated form. The default is OFF (messages are given in full).

STOP [{DISMOUNT|NODISMOUNT}]

All queued restore and copy requests are processed; then FS returns to the program which called it.

UPDATE tape-file[(version)] [newname] [pdn]
      [PAR={INIT|RESCAN},DOCUMENT,RECOVER]
UPDATE [(m)] [(n)] [pdn]
      [PAR={INIT|RESCAN},DOCUMENT,RECOVER]
UPDATE (m) [newname] [pdn]
      [PAR={INIT|RESCAN},DOCUMENT,RECOVER]

The UPDATE command copies the latest version of files on the FS tape to the copy tape.

VERIFY

The VERIFY command verifies the checksum information on the tape.

$cmnd

*FS command lines beginning with a "$" (except for $ENDFILE and $CONTINUE WITH) are executed as MTS commands.

Description: The above command descriptions give only a brief summary of *FS. The complete description is given in MTS Volume 19, Tapes and Floppy Disks.

January 1987

<u>*FTN</u>

Contents: The interface program for the FORTRAN-G and FORTRAN-H
                 compilers.

Use: The interface is invoked by the $RUN command.

Program Key: *FTN

Logical I/O Units Referenced:
                 SCARDS - source program (defaults to *SOURCE*).
                 SPRINT - compiler source program and object module list-
                          ings (defaults to *SINK* in batch mode).
                 SPUNCH - object module generated (defaults to *PUNCH* in
                          batch mode).
                 SERCOM - interface and compiler diagnostic error messages
                          (defaults to *MSINK*).
                 GUSER - responses to interface prompting messages
                          (defaults to *MSOURCE*).
                 0   - object module generated (if DECK option is
                          specified).
                 1   - edited source module generated (if EDIT option
                          is specified).

Return Codes: 0 - Successful return.
                 4 - Warning messages printed.
                 8 - Errors detected.
                 12 - Serious errors detected.
                 16 - Internal compiler error.

Description: The interface program for the FORTRAN-G and FORTRAN-H
                 compilers performs the following services:

                 (1) conversational entry and correction of the
                      parameters,
                 (2) intermixing of source modules, object modules, and
                      commands within the input stream,
                 (3) four source statement formats,
                 (4) availability of the source program reformatted to
                      standard IBM format,
                 (5) incorporation of the MTS line numbers in the source
                      listings and terminal diagnostics, and
                 (6) an object module format using CSI loader records
                      when possible to decrease loading time.

                 The complete description of the interface program is
                 given in MTS Volume 6, <u>FORTRAN in MTS</u>. A brief summary
                 of the interface options is given below.

Option	Default
BCD / NOBCD	NOBCD
COMMENT / NOCOMMENT	COMMENT
COND / NOCOND	COND
DECK / NODECK	NODECK
EDIT / NOEDIT	NOEDIT
EJECT / NOEJECT	EJECT
ERR / NOERR	ERR
EXPLAIN / NOEXPLAIN	NOEXPLAIN
ID / NOID	ID
LIB / NOLIB	NOLIB
LIST / NOLIST	NOLIST
LOAD / NOLOAD	LOAD
MAP / NOMAP	NOMAP
PRINT / NOPRINT	PRINT
OVER / NOOVER	NOOVER
QUIT / NOQUIT	See Vol. 6
SCAN / NOSCAN	NOSCAN
SM / NOSM	NOSM
SML / NOSML	NOSML
SOURCE / NOSOURCE	See Vol. 6
STRU / NOSTRU	NOSTRU
TEST / NOTEST	NOTEST
XL / NOXL	NOXL
XREF / NOXREF	See Vol. 6
CALIGN=n	CALIGN=0
CSHIFT=n	CSHIFT=0
FORMAT={LINE|IBM|EDITED}	FORMAT=EDITED
LINE=number	LINE=57
NAME=characters	NAME=MAIN
OPT={0|1|2|H|G}	OPT=G
OVER=parlist	None
SIZE=number	SIZE=4

January 1987

FTNTIDY

Contents: The FORTRAN "tidying" program.

Purpose: To tidy FORTRAN source programs into an easily readable
                 format and/or to produce cross-reference listings.

Use: The program is invoked by the $RUN command.

Program Key: *FTNTIDY

Logical I/O Units Referenced:
                 SCARDS - one or more FORTRAN source programs to be tidied
                          and/or cross-referenced.
                 SPRINT - the listing of the program(s) and the
                          cross-references.
                 SPUNCH - the tidied FORTRAN source program(s).
                 SERCOM - severe error comments.

Return Codes: 0 - Successful return.
                 4 - Warning messages printed.
                 8 - Error messages printed.
                 12 - Severe errors occurred.

Description: FTNTIDY may be used to tidy FORTRAN source programs
                 and/or to produce a cross-reference of the variables,
                 statement numbers, functions and subroutines, and the
                 FORTRAN logical I/O units used. By default, if the MTS
                 logical I/O unit SPUNCH is not assigned on the $RUN
                 command, FTNTIDY produces a listing of the source fol-
                 lowed by the cross-reference listings.

                 The complete description of the FTNTIDY program in given
                 in MTS Volume 6, FORTRAN in MTS.

January 1987

<u>*GASP</u>

Contents: The GASP simulation language. *GASP is a subroutine
                library file containing the modules for the GASPIIA
                simulation package.

Purpose: This file provides object modules to support simulation
                programs written in FORTRAN and organized for an event-
                oriented discrete simulation.

Use: The file should be concatenated to the file containing
                the main program on the $RUN command, e.g.,

                    $RUN main+*GASP

Program Key: *EXEC

Logical I/O Units Referenced:
                Two special GASP I/O units are referenced:

                    NPRNT - all output.
                    NCRDR - data input.

                NPRNT and NCRDR are set by the user.

Description: GASP is described as a FORTRAN-based discrete simulation
                language; in fact, GASP is a set of supporting subrou-
                tines that ease the writing of simulation programs in
                FORTRAN by providing certain statistical control and
                event-scheduling operations.   The user must provide his
                own main program, together with certain supporting sub-
                routines, which interact with and use the GASP routines.
                The version of GASP available in MTS in the public file
                *GASP is a modified version of GASPIIA; the original
                GASPIIA is well documented in the following reference:

                    <u>Simulation with GASP II</u>, by A. A. B. Pritsker and
                    Philip J. Kiviat, Prentice-Hall, 1969.

                Limitations of MTS GASP and the differences between MTS
                GASP and the GASP as described in the above reference are
                detailed in Computing Center Memo 246.   Users   are
                referred to the above for descriptions of GASP routines,
                their algorithms, and their calling sequences. Addition-
                al information may also be obtained via the MTS command
                $COPY W020:GASPINFO.

Example: $RUN *FTN
                    (User-supplied FORTRAN routines)
                $RUN -LOAD+*GASP
                    (User-supplied GASPIIA data)

                In the above example, the FORTRAN routines compiled
                by *FTN into the file -LOAD are executed with the
                GASP simulation package.

*GENDOC

Contents: The On-Line Documentation Facility.

Purpose: To allow users to produce MTS documentation in either
                terminal or hard-copy format.

| Use: The program has been deleted from the system.

*GPSSH2

| Contents: The GPSS/H (Version 2) system, a compiler and run-time
                support system for programs written in the GPSS/V pro-
                gramming language.

| Use: *GPSSH2 is invoked by the $RUN command.

  Program Key: *EXEC

  Logical I/O Units Referenced:
                SCARDS - source-program input to be compiled and
                         executed.
                SPRINT - standard output.
                GUSER - interactive debugging input.
                SERCOM - interactive debugging output; error and warning
                         messages.
                SPUNCH - update feature (optional) - "punched" copy of
                         updated source.
                1   - Jobtape #1.
                2   - Jobtape #2.
                3   - Jobtape #3.
                4   - SAVE file.
                5   - READ file.
                6   - update feature - master source file.
                7   - update feature - updated master source file.

| References: (1) GPSS/H User's Guide, 3rd Edition, by James
| O. Henriksen, Wolverine Software Corp.
| (2) Getting Started with GPSS/H, by Carson, Banks, and
| Sy, Wolverine Software Corp.
| (3) Simulation Using GPSS, by Thomas J. Schriber, John
| Wiley & Sons, 1974.
| (4) IBM General Purpose Simulation System V User's
| Manual, form SH20-0851.

  Description: GPSS/H is a new implementation of GPSS for IBM 370 and
                370-compatible computers. With a few exceptions, GPSS/H
                is upward compatible with IBM's GPSS/V. Most programs
                that run under GPSS/V can be run under GPSS/H without
                modification.   For a summary of the differences between
                GPSS/H and GPSS/V, please refer to Appendix C of the
| GPSS/H User's Guide ("Differences between GPSS/H and
                Other Versions of GPSS").

  Options: GPSS/H options are specified via the PAR field of the
                $RUN command invoking GPSS/H. In the case of conflicting
                specifications, the rightmost specification is used. In
                the descriptions that follow, allowable abbreviations are

indicated by underscoring; e.g., NOXREF indicates that NOXREF may be abbreviated as NOX.

SOURCE/NOSOURCE

If SOURCE is specified, a source listing of the program is written on SPRINT. SOURCE is the default for batch use. For terminal use, SOURCE is assumed if SPRINT is not assigned to the terminal, in which case NOSOURCE is assumed. If the program contains errors, erroneous lines and their associated error messages are always written on SPRINT.

XREF/NOXREF

XREF prints a cross-reference listing of all entity symbols and constants on SPRINT. The defaults for XREF/NOXREF are the same as those for SOURCE/NOSOURCE.

XSORT=A

By default, a cross-reference listing is grouped by entity classes and sorted by entity name/number within each class. Use of the XSORT=A option suppresses the grouping of a cross-reference listing into entity classes; i.e., the entire collection of entity names/numbers is sorted and printed without regard to entity class.

DICT/NODICT

The DICT option prints a dictionary of all GPSS entities used in a program. The dictionary is sorted by entity number and is useful for distin- guishing which entities have been used in a program and which are available for use. The NODICT option may be used to suppress printing of the entity dictionary. The defaults for DICT/NODICT are the same as those for SOURCE/NOSOURCE.

UPLIST/NOUPLIST

The UPLIST option requests that UPDATE specifica- tions be included in program output when the GPSS UPDATE feature is used. The NOUPLIST option may be used to suppress inclusion of an UPDATE listing. The defaults for UPLIST/NOUPLIST are the same as those for SOURCE/NOSOURCE.

ERR/NOERR

>   The ERR option echoes compilation error messages on
>   the terminal when the program compilation listing is
>   directed to a destination other than the terminal.
>   The NOERR option may be used to suppress the echoing
>   of compilation error messages.

WARN/NOWARN

>   The WARN option echoes compilation warning messages
>   on the terminal when the program compilation listing
>   is directed to a destination other than the termi-
>   nal. The NOWARN option may be used to suppress the
>   echoing of compilation warning messages.

LIST/NOLIST

>   The LIST option includes an object-code listing in
>   the program output. An object-code listing is of
>   use only to someone familiar with the internal
>   structure of GPSS/H. NOLIST is always the default
>   option.

XWF

>   The XWF option causes conditions which would normal-
>   ly merely result in the printing of an execution
>   warning message to be treated as fatal errors.   For
>   example, an attempt to store a value greater than
>   32767 in a halfword location ordinarily results in
>   the issuance of a warning message, and execution is
>   continued by storing a truncated value. In a very
>   large model, this action could result in a costly
>   run which produces meaningless results. Use of XWF
>   option prevents such disasters.

JTIF={V|360}

>   If Jobtapes to be read by GPSS/H have been written
>   by GPSS/V or GPSS/360, the JTIF=V or JTIF=360
>   options must be used, respectively.

JTOF={V|360}

>   If Jobtapes are to be written in formats compatible
>   with GPSS/V or GPSS/360, the JTOF=V or JTOF=360
>   option must be used, respectively. If neither of
>   these options are used, any Jobtapes written will be
>   written in GPSS/H format.

JTF={V|360}

> JTF=V is a convenient way of requesting both the JTIF=V and JTOF=V options. Similarly, JTF=360 requests both JTIF=360 and JTOF=360.

SIZE={A|B|C}

> The number of entities available to a GPSS program is determined in part by the SIZE option. SIZE=A, SIZE=B, or SIZE=C may be specified, with SIZE=A being the default. The explicit or implied use of a SIZE option establishes default allocations for the number of members of each GPSS entity class. The compiler may alter default allocations, based on its knowledge of the model being compiled. For example, if it sees that no entities of a given class are referenced, it will reduce the allocation of that entity class to zero. Allocations can also be automatically increased. For example, if the compiler encounters a reference to Facility number 350, it will guarantee that at least 350 Facilities are allocated for model execution. For a complete explanation, see the description of automatic reallocation of entities in Chapter 3 of the GPSS/H User's Guide.

> In many cases, the compiler cannot tell how many members of a given entity class are required. For example, if a model contains a "SEIZE PH1" Block, the number of Facilities required cannot be determined at compile time, because the values of PH1 are not known. For entity classes so referenced, the compiler has no alternative but to use the number of members implied by the SIZE option. If the number of members implied by the SIZE option is inadequate, one or more REALLOCATE control statements must be supplied to override the default allocation.

> The number of members of each entity class implied by SIZE=A, SIZE=B, and SIZE=C is summarized in tabular form in the description of the REALLOCATE statement in Chapter 6 of the GPSS/H User's Guide.

TEST

> The TEST option is used to invoke the GPSS/H interactive debugging system. When the TEST option is used, compilation and loading of a program proceed in the usual fashion; however, at the point at which program execution would normally begin, control is passed to the GPSS/H interactive debugging command processor, which types a READY message

on the terminal. At this point, the user assumes
(and ideally retains) control of program execution.
Commands then can be issued to selectively display
model output, to stop program execution at arbi-
trarily specified Blocks (breakpoints), and, if
necessary, to step through a program, one or more
Blocks at a time.

BIG

By default, the GPSS/H compiler keeps all of its
internal data structures in virtual memory. For
very large programs (thousands of statements), this
may result in excessive use of virtual memory. The
BIG option forces the compiler to make use of a
scratch file (named -$SCRATCH). When the BIG option
is used (presumably rarely in MTS), the file
-$SCRATCH should be explicitly created prior to
running GPSS/H, e.g., $CREATE -$SCRATCH SIZE=15OP.

NOSPIE

By default, the GPSS/H simulator traps all program
interrupts that occur during program execution.
When user-supplied FORTRAN or PL/I subroutines cause
program interrupts, determination of the exact cause
of the error may be made difficult by the automatic
trapping of the interrupt. When the NOSPIE option
is used, GPSS/H program interrupt trapping is disa-
bled, so that MTS system debugging tools, e.g., SDS,
can be used. NOSPIE is an acronym for "no set
program interrupt exit."

Return Codes: 0 - Successful return.
               4 - Error return.

| Examples: $RUN *GPSSH2 SCARDS=MODEL

In the above example, the source program is read
from the file MODEL. Source, cross-reference, and
dictionary listings are produced if run in batch
mode, or suppressed if run in terminal mode.

| $RUN *GPSSH2 SCARDS=MODEL PAR=SOURCE XREF DICT

The above example is the same as the previous
example, except that source, cross-reference, and
dictionary listings are always produced.

Using HELP and EXTERNAL Subroutines

The use of HELP and EXTERNAL subroutines is very simple in MTS.   In
order to use HELP or EXTERNAL subroutines, take the following steps:

(1) Code the GPSS program, invoking HELP routines with appropriate
     HELP Blocks and/or invoking EXTERNAL subroutines by means of
     appropriate declarations and references. The use of HELP blocks
     and EXTERNAL subroutines is described in Chapters 8 and 14 of
   | the GPSS/H User's Guide, respectively.
(2) Compile the HELP/EXTERNAL routines, using the appropriate
     compiler.
(3) Append the object-code file and any required libraries onto
     *GPSSH in a $RUN command.

In the example which follows, FORTRAN HELP/EXTERNAL subroutines con-
tained in the file MYFTN.S are compiled by the MTS FORTRAN/G compiler,
placing the resultant object code in the file -LOAD (by default).   The
$RUN command loads both GPSS/H and the HELP/EXTERNAL routines.

        $RUN *FTN PAR=S=MYFTN.S
    | $RUN *GPSSH2+-LOAD SCARDS=MYMODEL

Connecting Files and Devices to GPSS Programs

Extended I/0 capabilities introduced with release 1.0 of GPSS/H allow a
GPSS program to read and write files or devices directly, without use of
HELP subroutines, as was formerly necessary. The extended I/O capabili-
| ties of GPSS/H are described in Chapter 16 of the GPSS/H User's Guide.

Each I/0 statement type allows specifications of an input/output file or
device, by means of a FILE keyword.   In MTS, the FILE keyword is
supported as follows:

(1) If no FILE specification is given in an input statement, input
     is read from logical unit GUSER, which defaults to the job
     source stream for a batch run and to the terminal for an
     interactive run.
(2) If no FILE specification is given in an output statement, output
     is written to logical unit SERCOM, which defaults to the printer
     for a batch run and to the terminal for an interactive run.
(3) If "FILE=filename" is specified in an I/0 statement, the
     following procedure is used to associate "filename" with a file
     or device:

     (a) If "filename" starts with an alphabetic character and is an
          MTS logical I/O unit name, the file or device assigned to
          the logical I/O unit is used. The following example prints
          a one line message on SPRINT:

                  PUTPIC   FILE=SPRINT
              -YOUR PROGRAM IS IN TROUBLE

(Note carriage control in column 1.)

(b) If "filename" is of the form "UNITn", where "n" is an
     integer, the file or device assigned to logical I/0 unit
     "n" is used.   The "UNIT" prefix is necessary to maintain
     compatibility with IBM operating systems, under which file
     names cannot begin with a numeric character. The following
     example reads an integer value into fullword Savevalue X1
     from logical I/0 unit 8:

     GETLIST   FILE=UNIT8,END=EOFEXIT,ERR=BADINT,(X1)

(c) If "filename" satisfies neither of the above conditions,
     but is the name of an existing file, the file is read or
     written directly. Note that the GPSS/H naming conventions
     for files require that the name start with an alphabetic
     character and that the name be 1-8 characters long.   This
     restriction implies that temporary files and files belong-
     ing to user IDs other than the one under which GPSS/H is
     being run cannot be used. The following example reads a
     floating-point value into floating-point Savevalue XL$PARM
     from the file named MYDATA:

     GETLIST   FILE=MYDATA,END=EOFERR,ERR=BADFLOAT,(XL$PARM)

Interrupting Model Execution and Output

When the GPSS/H interactive debugging feature is used (PAR=TEST), model
execution may be interrupted at any point, by pressing the BREAK, ATTN,
PA1 (or similarly labeled) attention interrupt button of the terminal.
The attention interrupt button can also be used to terminate most forms
of GPSS/H output.   For example, if an interactive debugging DISPLAY
command produces too much output to be conveniently viewed on the
terminal, the output may terminated by pressing the attention interrupt
button. In such a case, a PRINT command can be issued routing output to
a more appropriate destination (such as a printer).

Transporting GPSS/H Programs and/or Data

If GPSS/H programs and/or data are to be transported to other sites, the
following precautions should be taken:

(1) Remember that by default, Jobtapes are read and written in
     GPSS/H format.   If Jobtapes are being prepared for use with
     IBM's GPSS/V or GPSS/360, GPSS/H format will be unacceptable.
     By using the appropriate PAR JTF or JTOF option, Jobtapes can be
     prepared for use with GPSS/V or GPSS/360.
(2) If the Jobtape file is being written on a magnetic tape for use
     under an IBM operating system, the following rules must be
     observed:

     (a) If the Jobtape is to be used under GPSS/V, the tape must be
          written with a logical record length of 416 and a blocksize

which is a multiple of 416.
- (b) If the Jobtape is to be used under GPSS/360, the tape must be written with an unblocked, fixed logical record length of 416.
- (c) If the Jobtape is to be used under GPSS/H and is written in GPSS/H format, the tape must be written with a record format of VBS, a logical record length long enough to contain the largest Transaction, and a reasonably large block size (for efficiency). Note that the largest any Jobtape Transaction can be is 2825 bytes, so a logical record length of 2825 will always work.

(3) GPSS/H READ/SAVE files can only be processed by GPSS/H. They are incompatible with those of other implementations of GPSS.
(4) If a READ/SAVE file is being written for use under GPSS/H on an IBM operating system, the tape must be written with a record format of U and a logical record length / block size of 4096.
(5) In all operations involving the copying of Jobtapes and READ/SAVE files, the @-TRIM I/O modifier must be used to avoid truncation of records.

Return Codes

At the completion of every run, GPSS/H passes a return code back to MTS. The return code can be interpreted as follows:

Value	Interpretation
0	Successful return
4	Syntactic error(s)
8	Error(s) in entity/symbol definition(s)
12	Semantic error(s)
16	Load/Control errors
20	Execution error
24	Output error
28	Invalid PAR option

Support of GPSS/H

GPSS/H is supported by the Wolverine Software Corporation. Questions and problems should be directed to

James O. Henriksen
Wolverine Software Corp.
| 4115 Annandale Road
| Annandale, VA 22003-2500
| (703) 750-3910

Users on the University of Michigan campus may also contact

Prof. Thomas J. Schriber
School of Business Administration
764-1398

*HPGLPLOT

Contents: A program to convert an MTS plot-file (PDS file) into a
                file containing HPGL (Hewlett-Packard Graphics Language)
                commands.

Use: The program is invoked by the $RUN command.

Program Key: *HPGLPLOT

Logical I/O Units Referenced:
                0   - plot-file to be translated.
| 23   - output plot file.
                SERCOM - program error messages.
                SPRINT - program nonplot output.

Parameters: The following parameters may be specified on the $RUN
                command. These will control the different options for
                the plot-file conversion. The minimum abbreviation for
                each parameter is underlined.

                CENTER

                        The CENTER option specifies that the plot is to be
                        centered on the page. The plot will be positioned
                        such that its midpoint is mapped to the page
                        midpoint, i.e., the point ((xmax-xmin)/2) of the
                        plot is mapped to the center of the page (this is
                        (5.5,4.25) in landscape mode and (4.25,5.5) in
                        portrait mode). The CENTER option will override any
                        SHIFT or ORIGIN option specifications. CENTER is
                        useful only if NOFIT is specified; the plot is
                        automatically centered in FIT mode.

                FIT

                        The FIT option optimally fits each plot onto the
                        plotter paper. The Hewlett-Packard 7550 Plotter
                        uses 8.5 x 11 inch paper with a drawing area on that
                        paper of 7.75 x 10.0 inches. A fitted plot will be
                        scaled so that it is as large as possible without
                        any part of the plot being clipped. FIT is the
                        default mode.

                FRAME=n

                        The FRAME option specifies which plot in the file is
                        to be converted, e.g., FRAME=5 will convert the
                        fifth plot. The default is FRAME=1. It is current-
                        ly not possible to convert the entire plotfile at

once. Since the HPGL page-eject command is plotter-
dependent, the MTS HP driver does not put any
page-eject commands in the output file. If all the
plots were converted at once, they would all be
drawn on the same sheet of paper, one on top of the
other.

LANDSCAPE

The LANDSCAPE option specifies that the plot con-
verted is to be oriented in landscape mode, i.e.,
the x-axis will be parallel to the 11-inch edge of
the paper.

NOFIT

The NOFIT option specifies that the coordinates to
be used in each plot are in absolute inches.   No
implicit scaling is performed on the plots, and
coordinates which are outside of the drawing area on
the plotter paper will be clipped appropriately.
The origin point (0.0,0.0) is at the extreme bottom
left-hand corner of a page, which means that coor-
dinates in the negative axis range will be clipped.

ORIGIN=(x,y)

The ORIGIN option may only be used in conjunction
with the NOFIT option. By default, the origin point
(0.0,0.0) of a NOFIT plot is located at the extreme
bottom left-hand corner of a page.   The ORIGIN
option allows the location (0.0,0.0) to be relocated
to an arbitrary point, not necessarily within the
boundaries of a physical page. It is up to the user
to ensure that a plot remains within the visible
region if this option is used.

Either the "x" value, the "y" value, or both may be
omitted from an ORIGIN command; unspecified parame-
ters will default to 0.0. A maximum of three digits
following the decimal point may be given.

The ORIGIN specification is not affected by the
presence of a SCALE specification (i.e., the origin
coordinates are not scaled). Using the effects of
these two options, it is possible to increase the
visible viewing region of a plot.

PEN(color,size)=newcolor

The PEN option respecifies the pen colors. "color"
and "size" specify the color and size of the pen to
be changed; "newcolor" specifies the new color of

the pen. The "size" parameter may be omitted if the
"color" parameter unambiguously specifies the pen to
be changed, e.g., there is not both a medium and
fine pen to the same color. For example, to have
the only red pen in the plot drawn as blue, specify
PEN(RED,)=BLUE; to have a fine black pen drawn as
green, specify PEN(BLAC,FINE)=GREE. If a PPEN re-
cord in a plotfile specifies a nonsupported color,
that color will be drawn as black.

PORTRAIT

The PORTRAIT option specifies that the plot con-
verted is to be oriented in portrait mode, i.e., the
x-axis will be parallel to the 8.5-inch edge of the
paper.

SCALE=scale-factor

The SCALE option specifies a scaling factor, which
must be in the range 0.0 < scale-factor ≤ 1.0. In
FIT mode, this scaling is applied to the image after
it has been fitted onto a page. The default scale
factor is 1.0.

SHIFT

The SHIFT option may only be used in conjunction
with the NOFIT option. By default, a plot produced
in NOFIT mode has its origin at the bottom left-hand
corner of a page. Specifying the SHIFT option will
cause each image to be translated up by one inch to
give more space at the bottom of the page in
landscape mode and more room along the left edge of
the page in portrait mode. The SHIFT option is the
same as specifying ORIGIN=(0.0,1.0) in landscape
mode and ORIGIN=(1.0,0.0) in portrait mode.

SLOW

The SLOW option plots the plot-file at a slower
speed than normal. This will help improve the
quality of the lines drawn, but will also increase
the amount of time taken to draw the plot.

Return Codes: 0 - Successful return.
               8 - Unit 0 was unassigned.
               12 - Plotfile conversion failed.

Description: This program reads in an MTS plot-file and translates it
              into HPGL commands using the *IG and the *IG.HPGLFILE
              device driver. The device driver will request a file

| name for the HPGL output if logical I/O unit 23 is not
| specified, and request the type of plotter to be used.

                   See Computing Center Memo 466, "*HPGLPLOT: Converting
MTS Plotfiles into Hewlett-Packard Graphics Language,"
for further information on conversion MTS plot-files to
HPGL plot-files.

Example: $RUN *HPGLPLOT 0=PLOTFILE

                   In the above example, the plot contained in the file
PLOTFILE is translated into an HPGL file.

January 1987

<u>*ICON</u>

Contents: The Icon programming language.

Use: The Icon processor is invoked by the $RUN command.

Program Key: *ICON

Logical I/O Units Referenced:
                 SCARDS - the Icon source program.
                 SPRINT - the program listing
                 SPUNCH - the Icon object module generated.
                 SERCOM - diagnostic and error messages.

Return Codes: Always zero.

Description: Icon is a programming language developed by Ralph Gris-
                 wold and others and is described in the following
                 references:

                     <u>The Icon Programming Language</u>, Ralph E. Griswold and
                         Madge T. Griswold, Prentice-Hall, N.Y., 1983
                     <u>A Guide to the Use of the Icon Book for Version 2
                         Users</u>, Ralph E. Griswold
                     <u>Reference Manual for the Icon Programming Language
                         Version 2</u>, Ralph E. Griswold and David R. Hanson,
                         TR 79-1a, Department of Computer Science, Univer-
                         sity of Arizona, Tucson, 1980
                     <u>Bibliography of the Icon Programming Language</u>, Ralph
                         E. Griswold

                 An Icon program is compiled by issuing the following $RUN
                 command:

                     $RUN *ICON SCARDS=source SPRINT=list SPUNCH=object

                 The resulting object program is executed by issuing the
                 following command:

                     $RUN object+*ICONLIB SCARDS=input SPRINT=output

Notes: The OPEN function currently only works with logical I/O
                 units, not files; for example, after 'F := OPEN("90")', F
                 is associated with SPUNCH. If the argument to OPEN is
                 not numeric, an obscure error message will be generated
                 unless 19 is assigned to a directory file, the format of
                 which is too obscure to describe here.

                 The I/O units of Icon have the following MTS equivalents:

nn	MTS Unit
90	SPUNCH
91	SCARDS
92	SPRINT
93	GUSER
94	SERCOM
0	0
..	..
19	19

*IF66, *IF77

Contents: The Interactive FORTRAN processors.

Purpose: To interactively compile, execute, and debug FORTRAN programs.

Program Keys: *IF66 and *IF77

Return Codes: Always zero.

Description: The Interactive FORTRAN processors are completely described in the section "Interactive FORTRAN" in MTS Volume 6, FORTRAN in MTS.

January 1987

<u>*IG</u>

Contents: A library of FORTRAN-callable subroutines for drawing and
              manipulating pictures.

Use: *IG should be concatenated to the file containing the
              user's object modules on the $RUN command, e.g.,

                  $RUN object+*IG

Program Key: *EXEC

Logical I/O Units Referenced:
              9   - CalComp plot file (only if a CalComp plot is
                       requested).
              25   - SCODL command file (only if the matrox PCR is
                       requested).

Description: The Integrated Graphics (IG) library is a set of FORTRAN-
              callable subroutines for drawing and manipulating pic-
              tures.   The interface to the user's program is device
              independent. A program that uses IG to perform its
              graphics can be run without alteration on any graphics
              terminal or on the CalComp plotter.

              IG also includes an interface from the *PLOTSYS library.
              This interface currently allows programs that produce
              CalComp plots to be run on a terminal with the plot
              displayed on the screen rather than being written to a
              plot file.

              For a complete description of the Integrated Graphics
              system, see MTS Volume 17, <u>Integrated Graphics System</u>.

January 1987

<u>*KERMIT</u>

Contents: The MTS implementation of the Kermit file-transfer
                protocol.

Purpose: To provide communications and file tranfer capabilities
                between MTS and another host, mini-, or microcomputers
                running Kermit.

 Use: MTS Kermit is invoked by the $RUN command. The computer
                to be communicated with must also be running the Kermit
                program.

Program Key: *KERMIT

Return Codes: 0 - Successful return.
                8 - Internal error.

Description: Kermit is a communications program that provides a
                mechanism for exchanging data between two like or unlike
                computers over standard telephone or computer network
                communications lines. Most commonly, Kermit is used
                between a host computer, such as the UM-MTS host or the
                UB-MTS host, and a microcomputer such as an Apple
                Macintosh or an IBM PC. The microcomputer uses its
                RS-232 serial communications port for data transfer.

                The Kermit protocol allows the transmission of both ASCII
                text and binary files over any network that will accept
                the printable ASCII characters. In the case of binary
                files, any characters that are non-printable are encoded
                into printable characters and decoded at the other end by
                the other Kermit. MTS Kermit can be used to send files
                between MTS and any other computer with the Kermit
                software.

                There are several microcomputer versions of the Kermit
                program which combine a terminal emulator with the Kermit
                file transfer mechanism. When a microcomputer is started
                using Kermit, it will display a prompt of the form
                "Kermit-xx>", where "xx" will be a number or name
                indicating the microcomputer being used. In the case of
                the IBM PC, this name is "MS" indicating that the
                microcomputer uses the MS-DOS operating system.

Documentation: Computing Center Memo 473, "Kermit on MTS," describes the
                MTS implementation of the Kermit program. Computing
                Center Memo 904, "Using the Macintosh as a UMnet Termi-
                nal: The MacKermit Program," and Computing Center Memo
                922, "Using the PC-Compatible as a UMnet Terminal: The

Kermit-MS Program," describe the Macintosh and IBM PC
versions of Kermit. The "Kermit Users Guide", a lengthy
document published by the Columbia University Center for
Computing Activities, where Kermit originated, is also
available.   The guide provides a complete description of
several versions of Kermit that run on a variety of
systems, but is not MTS specific. This document resides
in the MTS file PC:KERMIT.GUIDE and can be printed on the
line or page printers; it is 260 pages long.   Another
document is the "Kermit Protocol Guide," which describes
the internal workings of the Kermit communications
scheme. This document will only be useful to programmers
who are implementing new versions of Kermit. It is in
the file PC:KERMIT.PROTO and is 94 pages long.

Examples: The example below illustrates the transfer of a character
                 (text) file from MTS to the IBM Personal Computer:

                 $Run *Kermit
                 KERMIT-UM> MTS KERMIT (UM) Version 1.31 (16:41 Sep12/86)
                 KERMIT-UM>SEND TEST-ASC
                 KERMIT-UM>Preparing to send file 'TEST-ASC'.
                 KERMIT-MS>RECEIVE

                 (... transfer starts, runs, and completes ...)

                 KERMIT-MS>CONNECT
                 KERMIT-UM>STOP

                 The example below illustrates the transfer of a character
                 (text) file from the IBM Personal Computer to MTS:

                 $Run *Kermit
                 KERMIT-UM> MTS KERMIT (UM) Version 1.31 (16:41 Sep12/86)
                 KERMIT-UM>REC PC-FILE
                 KERMIT-UM>File being received to be placed into MTS file
                 KERMIT-MS>SEND TEXTFILE

                 ... transfer starts, runs, and completes ...

                 KERMIT-MS>CONNECT
                 KERMIT-UM>STOP

<u>*LABEL</u>

Contents: The MTS magnetic tape-initializing program.

Purpose: To initialize a labeled or unlabeled magnetic tape.

Use: The program is invoked by the $RUN command.

Program Key: *LABEL

Logical I/O Units Referenced:
                 GUSER - input from the user.
                 SERCOM - prompting for input and error comments.

Return Codes: Always zero.

Description: The program prompts for the pseudodevice name of the tape
                 to be initialized, the density (200, 556, 800, 1600, or
                 6250) at which it is to be initialized, the volume name,
                 and an optional owner ID.   This tape must have been
                 mounted with RING=IN specified on the mount request. If
                 the last two parameters (volume name and owner ID) are
                 omitted, the tape is initialized as an unlabeled tape,
                 i.e., six filemarks are written at the specified density
                 at the beginning of the tape and the tape is rewound. If
                 the volume name is given (1 to 6 characters without
                 embedded blanks or commas), the tape is initialized as a
                 labeled tape, is rewound, and label processing in
                 enabled. The specified volume name must then be used for
                 all future mounts of the tape. An optional owner ID (the
                 fourth parameter) will be included in the volume label if
                 given. It must not be longer than 10 characters.

                 If a volume label is given, the tape will be labeled
                 according to the IBM OS/VS magnetic-tape labeling stand-
                 ard, which is the standard in MTS.   However, if the
                 keyword parameter LBLTYPE=ANSI is specified as the last
                 parameter (after the owner ID, or volume name if the
                 owner ID is omitted), the program will write the volume
                 label using the American National Standards Institute
                 (ANSI) format. The tape should be ANSI-labeled if it is
                 to be exchanged with a computer installation that does
                 not support the IBM standard, but does support the ANSI
                 standard.

| The program will then wait for another set of parameters
                 to initialize a second tape. An end-of-file will termi-
                 nate the program.

Examples: $MOUNT C0000 9TP *TAPE* RING=IN 'TAPE ID'
                 $RUN *LABEL
                 TAPE 1600 0039JR UNIVOFMICH
                 $ENDFILE

                         In the above example, the tape *TAPE* is initialized
                         to 1600 bpi. The tape is labeled with a volume name
                         0039JR and an owner ID UNIVOFMICH.   The IBM OS/VS
                         labeling standard is used.

                 $MOUNT C0001 7TP *T* RING=IN 'TAPE ID'
                 $RUN *LABEL
                 T 556
                 $ENDFILE

                         In the above example, the tape *T* is initialized at
                         556 bpi. The tape is unlabeled.

                 $MOUNT C0002 9TP *** RING=IN 'TAPE ID'
                 $RUN *LABEL
                 *** 1600 DATA81 LBLTYPE=ANSI
                 $ENDFILE

                         In the above example, the tape *** is initialized to
                         1600 bpi.   The tape is labeled with a volume name
                         DATA81 using the ANSI-standard label type.   The
                         owner ID field will be blank.

January 1987

<u>*LABELS</u>

Contents: A mailing-label formatting program.

Purpose: To format data for multiple-label-per-sheet label stock.
               This is most commonly done with names and addresses, but
               other types of text may be formatted. Addresses may be
               sorted by zip code.

Use: This program is invoked by the $RUN command.

Program Key: *LABELS

Logical I/O Units Referenced:
               SCARDS - text to be formatted into labels. The input may
                       be in one of three formats; see the description
                       of the FORMAT option.
               SPRINT - labels formatted for printing. If the labels
                       are to be printed on the Xerox 9700, this will
                       be the input to *PAGEPR.
               SERCOM - error messages.

Options: The PAR field contains options that specify the format of
               the input, the layout of the label stock, how many copies
               of each label are to be printed, whether succeeding
               labels are printed across or down the page, and whether
               zip-code sorting is desired.   The options are:

               FORMAT={FREE|LINE|(i1-j1,i2-j2,...)}

                     The FORMAT option specifies the format of the input
                     from SCARDS. For FORMAT=FREE, the text for each
                     label begins on a new line. Text for successive
                     lines is delimited by semicolons (the separator
                     character may be changed by the DELIMITER option).
                     To continue the text for a label onto another line,
                     begin that line with a plus "+". The default is
                     FORMAT=FREE.

                     For FORMAT=LINE, the text for each line of a label
                     begins on a new line. The first column of the first
                     line for each label should contain a "1". The first
                     column of succeeding lines is ignored. Any line
                     beginning with a "1" begins a new label, and the
                     text for a label always begins in column 2. The
                     text for one label line may not be continued from
                     one input line to another.

                     FORMAT=(i1-j1,i2-j2,...) specifies a "fixed" format
                     for input. The text for each label begins on a new

input line. The text for the first line of the
label is in columns i1 through j1, the text for line
two is in columns i2 through j2, etc.

DELIMITER='c'

"c" will be used as the separator for lines in
free-format input. The default value for "c" is the
semicolon ";".

[NO]ZIP[=n]

The ZIP option specifies that labels are to be
printed in zip-code order, and that the zip code is
contained in the last line of each label.   ZIP=n
specifies that the zip code will be found on the nth
line of each label. Zip codes may be either 5- or
9-digit; 9-digit zip codes must have the form
"iiiii-iiii". If the input contains a mixture of 5-
and 9-digit zip codes, the 5-digit zip codes are
sorted as if they were iiiii-0000.   The zip code
must be the last non-blank item on the line.

NOZIP (the default) specifies that the input is not
to be sorted, but is to be printed in the order
input.

ORDER={ACROSS|DOWN}

The ORDER option specifies the order in which labels
appear on the page. For ORDER=ACROSS, succeeding
labels are printed across a row, and then down to
the next row.   For ORDER=DOWN, labels are placed
down the first column, then down the second, etc.
For ORDER=ACROSS, the second label printed on the
page is beside the first; for ORDER=DOWN, it is
below the first. The default is ORDER=ACROSS.

COPIES=n

"n" copies of each label will be printed, one
immediately following the other.   To produce "n"
sets of labels, use the output from *LABELS "n"
times.

WRAPAROUND={ON|OFF}

If the text for one line of a label is too long to
fit on the label, WRAPAROUND=OFF specifies that the
text should be truncated at the line length.   If
WRAPAROUND=ON is specified, the text will be con-
tinued onto the next line of the label.   The
continued line will be indented two spaces. Note

that this process may produce more lines than will
fit on the label, in which case the last line(s)
will be lost. The default is WRAPAROUND=ON.

PAPER={LABEL24|LABEL33}

The PAPER option specifies the name of a predefined
label stock format.   The stock format may also be
defined (see below). Two types of stock are cur-
rently defined.   PAPER=LABEL33 specifies 33 labels
per 8-1/2" x 11" sheet, 3 across by 11 down.   Each
label is 2-13/16" wide by 1" high. PAPER=LABEL24
specifies 24 labels per 8-1/2" x 11" sheet, 3 across
by 8 down. Each label is 2-13/16" wide by 1-3/8"
high.   For both types of labels, printing at 10
characters and 6 lines per inch is assumed.   The
default is PAPER=LABEL33.

TM, VG, LD, LM, LW, HG, VL, HL

These options are used to describe a user-defined
label stock arrangement. See the description of
these options below.

Return Codes: Always zero.

Description: <u>User-Defined Forms</u>

8 parameters are used to describe the size of a label and
the arrangement of labels on a sheet. The units of these
parameters are characters and lines rather than inches.
So, when defining a form, both the label stock itself and
the character size and spacing used by the printer must
be considered. The parameters are:

TM=l1   Top margin, the number of lines between the top
         of the sheet and the first line of a label.
         *LABELS uses a ":" carriage control to start
         each page, then spaces down l1 lines to the
         first label.   If l1=0, printing will begin on
         the top line. On some devices, this will result
         in the first line of text being "cut off" at the
         top.

LM=c1   The number of characters in the left margin
         before the first label.

LW=c2   The width of a label, in characters. For some
         devices, including the Xerox 9700, printing
         across the full width of the label will cause
         the first and/or last character to "bleed" off
         the edge.   For such devices, it is better to
         define the label width to be one or two charac-

ters less than the actual width of the label,
and use a larger-than-actual LM and HG.

LD=l2   The depth of a label, in lines. This is the
number of lines of text than can be printed on
each label. *LABELS begins printing at the top
of each label. For some devices, including the
Xerox 9700, printing the full depth causes a
line to be "cut off" at the top or bottom. For
such devices, setting l2 to one less than the
actual depth, and setting VG to one larger than
actual will fix the problem.

HG=c3   Horizontal gap between labels, in characters.
For most label stock, this is zero, but see the
description of LW.

VG=l3   Vertical gap between labels, in lines. For most
label stock, this is zero, but see the descrip-
tion of LD.

HL=n1   The number of labels across a sheet
(horizontally).

VL=n2   The number of labels down a sheet (vertically).

TM, LM, VG, and LG default to 0. LW, LD, HL, and VL must
be specified. These parameters may not be used to
"override" the specifications of the predefined forms
LABEL33 and LABEL24.

The accompanying diagram illustrates these parameters:

For the predefined forms LABEL33 and LABEL24, the values
of these parameters are:

	TM	LM	LD	LW	HG	VG	HL	VL
LABEL33	1	0	5	28	0	1	3	11
LABEL24	1	0	7	28	0	1	3	8

<u>Printing Labels on the Xerox 9700</u>

PAPER=LABEL33 and PAPER=LABEL24 will produce output suit-
able for papers LABEL33 and LABEL24, respectively, assum-
ing the font used prints 10 characters and 6 lines per
inch. For example,

```
 $R *LABELS SCARDS=ADDR SPRINT=-T PAR=PAPER=LABEL33
 $R *PAGEPR SCARDS=-T PAR=PORTRAIT,PAPER=LABEL33,MARGIN=0
```

will work. If a font with different spacing (e.g., 8
lines and 13.6 characters per inch) is used, a user-
defined form must be used.

January 1987

```
          ←LM→←————————————HL————————————→
         ┌─────────────────────────────────────────┐
TM  ↑    │                                         │
    ↓    │                                         │
    ↑    │  ┌───────────┐   ┌───────────┐   ┌───────────┐
    │    │  │    LW     │HG │           │   │           │
    │    │  │←─────────→│←─→│           │   │           │
    │    │  │           │   │           │   │           │
    │    │  └───────────┘   └───────────┘   └───────────┘
    │    │                    ↑ VG
    │    │                    ↓
    │    │  ┌───────────┐   ┌───────────┐   ┌───────────┐
    │    │  │           │   │ ↑         │   │           │
VL  │    │  │           │   │ │ LD      │   │           │
    │    │  │           │   │ ↓         │   │           │
    │    │  └───────────┘   └───────────┘   └───────────┘
    │    │
    │    │
    │    │
    │    │      ...           ...           ...
    │    │
    │    │
    │    │
    │    │      ...           ...           ...
    ↓    │                                         │
         └─────────────────────────────────────────┘
```
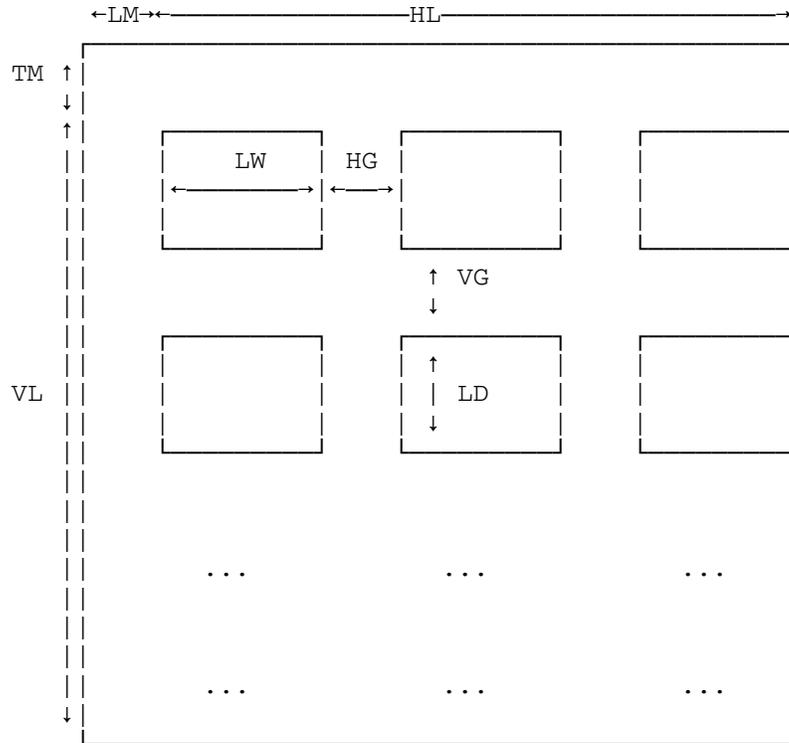
*LABELS does not support proportional fonts.

<u>Zip-Code Sorting</u>

The algorithm used to locate zip codes within a line is
intended to minimize the chances of "finding" something
other than a zip code. It will miss some zip codes. For
example, the zip code in

        Ann Arbor, MI 48109

will be found, but the zip code in

        Ann Arbor, MI 48109 USA

will be missed, because the zip code must be at the end
of the line. Any label in which a zip code is not found
will appear at the beginning of the output.

<u>Error Conditions</u>

If conflicting parameters are specified, or if a user-
defined form is incompletely specified, *LABELS will
print an error message and terminate without processing
any input data. If a line must be truncated to fit on a
label, an error message will be printed. If the text for

a label requires more lines than will fit on a label, a message will be printed, and the last line(s) for that label will be omitted.

Example: The example below shows the same label data in each of the three formats:

FORMAT=LINE

```
   1John Doe
    123 Main St.
    Ann Arbor, MI  48104
   1James Brown
    2234 State St.
    Ann Arbor, MI  48108
   1Mary Smith
    5224 Woodward
    Detroit, MI  48202
   1Jane Doe
    2214 S. Woodward
    Birmingham, MI  48011
   1Paul Harris
    3445 Hunter
    Birmingham, MI  48009
```

FORMAT=FREE

```
   John Doe;123 Main St.;Ann Arbor, MI  48104
   James Brown;2234 State St.;Ann Arbor, MI  48108
   Mary Smith;5224 Woodward;Detroit, MI  48202
   Jane Doe;2214 S. Woodward;Birmingham, MI  48011
   Paul Harris;3445 Hunter;Birmingham, MI  48029
```

FORMAT=(1-12,13-29,30-60)

```
   John Doe    123 Main St.      Ann Arbor, MI  48104
   James Brown 2234 State St.    Ann Arbor, MI  48108
   Mary Smith  5224 Woodward     Detroit, MI  48202
   Jane Doe    2214 S. Woodward  Birmingham, MI  48011
   Paul Harris 3445 Hunter       Birmingham, MI  48009
```

January 1987

*LABELSNIFF

Contents: The tape-identification program.

Purpose: To print tape information in an intelligible format.

Use: The program is invoked by the $RUN command.

Program Key: *LABELSNIFF

Logical I/O Units Referenced:
                 SPRINT - tape information output.
                 SERCOM - the message "ENTER TAPE NAME" as well as confir-
                          mations and error comments.
                 GUSER - input for tape names, if the PAR field is
                          omitted or if unit 0 is not specified.
                 0   - tape to be scanned.

Parameters: The following parameters may be specified in either the
                 PAR field of the $RUN command or in any GUSER input line.
                 The parameters must be separated by a blank.

                 TAPE=xxx   The name of the tape to be scanned, e.g.,
                            TAPE=*T*, if unit 0 is not specified.

                 FILES=xxx The number of data sets on the tape to be
                            scanned, e.g., FILES=10. If this parameter is
                            omitted, all of the data sets will be scanned.

                 FS / NOFS FS enables the automatic recognition and pro-
                            cessing of *FS tapes.   NOFS disables this
                            processing. The default is FS.

                 LP / NOLP If LP (label processing) is specified, the
                            tape labels will be read by the system tape
                            routines and the tape information will be
                            passed to *LABELSNIFF for interpretation. If
                            NOLP is specified, *LABELSNIFF will attempt to
                            read and interpret the label information. The
                            default is LP. Pool tapes will always be read
                            by the system tape routines.

                 OS / NOTOS If OS is specified, the LRECL (logical record
                            length) in V and D record formats is incre-
                            mented by 4. If NOTOS is specified, the LRECL
                            remains consistent with the MTS tape conven-
                            tions. The default is NOTOS.

                 POSN=xxx   This overrides the rewinding of the tape
                            before the processing starts.   The tape is

positioned to the beginning of the specified data set, if possible. Otherwise, the tape will be rewound. Data set numbers or names may be entered immediately after the equal sign, e.g., POSN=*2* or POSN=COUNT.

REW   The tape is rewound to the beginning of the tape after the processing is completed (the default).

NOREW   The tape is positioned at the logical end of the tape or after the last data set scanned. If an attention interrupt occurs, and if the user wishes to discontinue, the tape is positioned just before the current data set.

ENGLISH   The approximate tape length is given in feet (the default).

METRIC   The approximate tape length is given in meters.

SKIP   All data blocks are skipped. The program does not provide vital information such as record count, tape length, and average and maximum block lengths.

SHORT   A short column format, with only data set name, file number, block count, record count, tape length, and record format, is provided.

LONG   A long column format provides in addition to the short column format described above, average and maximum block lengths, creation date, expiration date, tape mode, user signon ID, and batch receipt number.

              The default for SHORT or LONG depends on the maximum output length of the logical I/O device unit SPRINT.

Return Codes: 0 - Successful return.
              4 - Minor errors encountered.
              8 - Nonfatal tape error encountered.
            12 - Fatal tape error encountered.
            16 - System error encountered.

Description: The complete description of *LABELSNIFF is given in MTS Volume 19, Tapes and Floppy Disks.

January 1987

<u>*LALRGEN</u>

Contents: The LALRGEN program.

Purpose: To determine whether a context-free grammar is LALR(1)
                 [look-ahead LR(1)].

Use: The program is invoked by the $RUN command.

Program Key: *LALRGEN

Logical I/O Units Referenced:
                 SCARDS - control cards and grammar to be processed.
                 SPRINT - the results of the grammar test including diag-
                          nostic messages, listings, and other user-
                          requested information.
                 SPUNCH - the generated parse tables.
                 SERCOM - diagnostic messages.

Return Codes: 0 - Successful return.
                 4 - Warning messages printed.
                 8 - Errors detected.

Description: LALRGEN is a program that tests whether or not a
                 context-free grammar is LALR(1) [look-ahead LR(1)], and
                 optionally punches a deck of tables suitable for driving
                 an LALR(1) parser.   The program was written in 1976 by
                 Thomas G. Szymanski of the Department of Electrical
                 Engineering and Computer Science, Princeton University
                 and subsequently adapted for use in MTS.

                 This description assumes the reader is familiar with LALR
                 parsing. For background, see Aho and Ullman, <u>Principles
                 of Compiler Design</u>, Addison-Wesley (1977).

                 The program is run with the MTS $RUN command using the
                 four MTS logical I/O units SCARDS, SPRINT, SPUNCH, and
                 SERCOM.   The program reads the grammar to be processed,
                 as well as certain control cards from SCARDS.   The
                 program writes the results of the grammar test on SPRINT,
                 including diagnostic messages, listings, and other user-
                 requested information. The program also writes diagnos-
                 tics on SERCOM if it is assigned to a file or device
                 different from SPRINT.   The program writes the parse
                 tables on SPUNCH. The format and quantity of output on
                 SPRINT and SPUNCH can be controlled by several methods
                 that are described in "Printed Output" and "Punched
                 Output" below. The usual MTS defaults apply if SCARDS,
                 SPRINT, SPUNCH, or SERCOM are not assigned. The follow-
                 ing is a prototype run command for LALRGEN:

```
$RUN *LALRGEN SCARDS=grammar SPRINT=listing
SPUNCH=parser-tables PAR=options
```

Input:

The input to LALRGEN consists of the PAR field options
and three kinds of cards.   The first card of the input
deck is called the "size parameter" card, and the rest of
the input consists of interspersed "option" cards and
"production" cards. The maximum length of any input line
to LALRGEN is 256 characters.

PAR Field Options

The PAR field options for the $RUN command include all of
the options available on the @PRINT, @PUNCH, and @DEBUG
option cards. These have the same effect in the PAR
field as on the option cards and may be specified in
either place. Two additional options, PRINT and NOPRINT,
may be specified in the PAR field to control the amount
of output produced on SPRINT.   The PRINT option is
discussed below under "Printed Output."   The PAR field
options may appear in any order and must be separated by
one or more blanks. The length of the PAR field is
limited to 256 characters.

Size Parameter Card

The first card of the input deck contains a sequence of
six integer size parameters to the program. The integers
must be separated by one or more blanks, and they must
appear in the order given below. All six size parameters
control the allocation of storage for holding the grammar
and the intermediate results of the computations on it.
The six size parameters are:

   (1) MAX_#_SYMS, an upper bound on the number of
        symbols in the vocabulary of the grammar,
   (2) MAX_#_PRODS, an upper bound on the number of
        productions in the grammar,
   (3) MAX_GRAMMAR_SIZE, an upper bound on the total
        number of symbols in all the productions of the
        grammar,
   (4) MAX_#_STATES, an upper bound on the number of
        states that the generated parser can contain,
   (5) MAX_#_KR_ITEMS, an upper bound on the number of
        LR(0) items in all of the basis sets of the
        states of the parser,
   (6) MAX_#_AFFECTED, the size of an internal table.

A value of 500 for MAX_#_AFFECTED should be sufficient
for most grammars. MAX_#_SYMS, MAX_#_PRODS, and MAX_#_S-

TATES may not exceed 999. If they are given larger values, 999 is used instead.

Production Cards

A symbol in the vocabulary of the input grammar may be any of the following: (1) the single character "<", (2) a string of up to 20 nonblank characters not starting with a "<", or (3) a string of up to 20 characters starting with a "<", followed by a nonblank character, followed by anything up to and including the next ">". The special character "@" may never appear anywhere on a production card. Symbols may actually be longer than 20 characters, but the program simply ignores all but the first 20 characters of any such symbol.

Productions are punched one per card with one or more blanks between symbols. A symbol is assumed to be a left-hand side if it begins in column one. Any other symbols on the card are assumed to be the right-hand side of the production. If column one is blank, the symbols on the card are assumed to be the right-hand side of a production with the same left-hand side as the preceding card. Lambda productions are legal, and are represented by a card with no right-hand side symbols. This implies that a blank card is a meaningful production card.

The first production card must contain an "augmenting" production that has the following characteristics. First, the left-hand side must be the start symbol of the grammar, and must not appear elsewhere in the grammar. Second, the rightmost symbol in the right-hand side must be a terminal symbol that does not appear elsewhere in the grammar.

A grammar symbol is assumed to be a terminal symbol if it never appears as the left-hand side of a production. Similarly, any symbol that appears on the left-hand side of a production is assumed to be a nonterminal. The program makes these assumptions whether or not the symbol contains the angle brackets "<" and ">".

Option Cards

Option cards are instructions to LALRGEN to perform specific operations. Option cards begin in column one with an "@" immediately followed by the option name. Usually option cards may appear anywhere after the size parameter card. Descriptions of the five types of option cards follow.

@NUMBER [symbol] ...

              This card is used for controlling the numbers
         assigned to terminal and nonterminal symbols of the
         grammar. This option is useful for interfacing a
         lexical analyzer with a parser for a grammar that is
         likely to be changed in the near future. Terminal
         symbols are normally assigned ascending numbers
         (starting with one) in order of their occurrence in
         the grammar. The @NUMBER option card allows a
         terminal to "occur" before its first use in the
         grammar, thus obtaining a number other than the one
         that would normally be assigned to it. Consider the
         following sequence of input cards:

              @NUMBER tart strudel
              @NUMBER cherry
              <goodie> <flavor> <kind>
              <flavor> apple
                       cherry
              <kind> pie
                     strudel
              tart
              @END

         This results in the following assignment of numbers
         to terminals:

              tart   1
              strudel   2
              cherry   3
              apple   4
              pie   5

         Notice that an @NUMBER card may contain more than
         one symbol.

         Nonterminals are also assigned consecutive numbers.
         These numbers start after the number of the last
         terminal symbol, but they are assigned in reverse of
         the order in which the nonterminals occur in the
         grammar. The @NUMBER card can force the occurrence
         of a nonterminal before its first use in the
         grammar, but the numbers assigned in this way are
         consecutive, decreasing integers starting with the
         total number of symbols in the vocabulary.

    @END

         This card is used to indicate the end of the
         grammar.   Any cards after the @END are ignored by
         LALRGEN. If it is missing, it will automatically be
         inserted by the program.

@PRINT [LASET] [CLOSURE]

This card causes the program to print more than the default information about the generated parser (see the subsection "Printed Output" below for an explanation of the PRINT option). LASET causes the lookahead set of each LR(0) item to be printed when the item is printed. CLOSURE causes the listing of the items in each state to include the closure items as well as the basis items. CLOSURE is expensive in terms of both time and paper. The options may appear in any order.

@PUNCH [ALGOLW] [INTEGERS] [XPL] [EXPL] [BCPL]
    [UNPACKED_ALGOLW] [UNPACKED_INTEGERS] [UNPACKED_XPL]
    [UNPACKED_EXPL] [VOCABULARY]

This card directs the program to punch the tables with which an LALR(1) parser can be driven. Each option designates a type of table, and any combination of tables may be punched in one run of LALRGEN. XPL and EXPL are synonymous, as are UNPACKED_XPL and UNPACKED_EXPL. The different table formats are described below. If an @PUNCH card does not appear in the LALRGEN input, no tables are punched, and the logical I/O unit SPUNCH is not used. The options may appear in any order.

@DEBUG [FNA] [PROP] [MOVES] [CLOSURE1] [CLOSURE2]

This card causes the printing of various kinds of debugging information. This option card is normally used only by people modifying LALRGEN itself. The options may appear in any order.

Output:

Conceptually, the output of LALRGEN is a description of the states and functions (tables) that are needed to do an LR parse of the input grammar. The program produces this output in several forms; it writes the tables in a form suitable for human reference on SPRINT, while it writes copies of the tables in a variety of forms suitable for direct incorporation into a parser on SPUNCH.

The tables that LALRGEN produces do not duplicate exactly the form given in the literature with separate "parsing action" and "go to" functions. Instead, the program constructs one function that encodes both the action and the next state in a single numeric value. We will call this function trans. In the following paragraphs, we will describe how to extract the usual action and goto

functions from <u>trans</u>, and how <u>trans</u> is represented in the various outputs of LALRGEN.

LALRGEN assigns integer values to the vocabulary symbols, the productions, and the parser states. <u>trans</u> is actually a function on integers, but for simplicity we will say "state s" when we mean "the integer assigned to state s," and the analog for other grammar objects. This minor ambiguity should be compensated by the resulting simplification of the exposition.

Like <u>action</u> and <u>goto</u>, the domain of <u>trans</u> is S x V, where S is the set of parser states (generated by LALRGEN) and V is the set of vocabulary symbols. The range of <u>trans</u> is the integers interpreted as follows, where "s" is a state and "v" is a symbol:

```
action(s,v) = shift                    if trans(s,v) > 1
              accept                   if trans(s,v) = 1
              error                    if trans(s,v) = 0
              reduce by -trans(s,v)    if trans(s,v) < 0

goto(s,v)   = error                    if trans(s,v) ≤ 1
              trans(s,v)               if trans(s,v) > 1
```

The start state of the parser is always state 1. Because of the restrictions on the augmenting production, acceptance can be and is reported at the time that the last terminal symbol would have been shifted. For this reason, reduction to the start symbol is never actually performed, and the true final state of the parser is never generated by LALRGEN. The advantage of this approach is that actions on the empty string do not need to be computed.

Clearly, one way to represent the function <u>trans</u> is to print a large matrix with one entry for each possible state-symbol pair. LALRGEN can do this, but for a large grammar this matrix can be huge. An alternate representation is therefore provided, called "packed," that avoids explicitly listing all the zeros (errors) inevitably present in the "unpacked" table. Packed tables consist, for each parser state, of a "symbol list," a parallel "action list," and a "default action." To find the value of <u>trans</u>(s,v) we look for "v" in the symbol list of "s". The corresponding value in the action list is the value of <u>trans</u>(s,v). If "v" is not in the symbol list for "s", then the value of <u>trans</u>(s,v) is the default action for "s".

Printed Output

The printed output of LALRGEN appears primarily on
SPRINT. Error messages are duplicated on SERCOM if it is
assigned to a file or device different from SPRINT.   The
entire printed output except for error messages is
suppressed if the PRINT option is off.   This supersedes
any @PRINT or @DEBUG option cards that may request
printed output. The PRINT option defaults off unless

   (1)   the run is made from batch,
   (2)   SPRINT is explicitly assigned on the $RUN com-
         mand, or
   (3)   an @PRINT or @DEBUG option is specified in the
         PAR field.

The PRINT option may be explicitly set on or off by
specifying PRINT or NOPRINT in the PAR field.

The SPRINT output of LALRGEN begins with a listing of the
input cards as they were read. The size parameter card
is omitted from this listing. Option cards are listed,
and any errors detected in them are reported here.
Following this is a listing of the productions and their
assigned numbers. The productions are formatted with the
usual punctuation for context-free grammars. Following
the productions is a listing of the vocabulary symbols
and their assigned numbers, and a listing of any non-
terminals that derive the empty string.   Next is a
listing of any conflicts encountered during construction
of the LALR(1) parser.   The longest component of the
printed output is a detailed listing of the results of
the LALR(1) analysis. This includes the LR(0) items that
comprise each state, and the packed parser tables.   The
last item printed by LALRGEN is a table of statistics
that, among other things, shows the values given on the
size parameter card, and the values actually required.
This information can be used to economize on storage
space for runs on similar grammars.

The detailed listing of the results of the analysis
mentioned above deserves more comment.   This listing
proceeds state by state in the order that the states were
generated (and numbered).   After the state number, the
program prints the LR(0) items that form the basis of the
state, and after this, depending on the use of the @PRINT
card, the program prints the remaining LR(0) items in the
state (the closure). If look-ahead sets are requested,
the program will print a list of terminal symbols after
each LR(0) item. These symbols will be the ones that can
appear after the right-hand side of the item in a legal
input string.   The next item printed for each state is
the portion of the packed parser tables that is associ-

ated with the state. The symbol list and action list for the state are merged under the heading "MOVES:" with a sequence of phrases of the form "s ON v." In such a phrase, "s" is the value on the action list corresponding to the symbol "v" on the symbol list. Following all the "moves" is the default action under the heading "DE-FAULT:". The last item printed for each state is a kind of cross-reference information that is not strictly part of the parser tables. Starting with the header, "AC-CESSED BY:", this line is a list of all the states that can cause transitions to this state.

Punched Output

Seven distinct kinds of tables can be punched by LALRGEN. They are described here under the @PUNCH options that request them.

UNPACKED_ALGOLW

> For this kind of table, LALRGEN punches six ALGOLW function procedures. NUM_PRODUCTIONS, NUM_STATES, and NUM_SYMBOLS take no parameters and return the number of productions, the number of states, and the number of vocabulary symbols, respectively. LHS_-SYMBOL and RHS_LENGTH each take a single integer parameter, a production number. LHS_SYMBOL returns the assigned number of the symbol on the left-hand side of the production. RHS_LENGTH returns the number of symbols on the right-hand side of the production. TRANS takes two integer parameters, a state number and a symbol number, and returns the corresponding value of the function <u>trans</u>.

UNPACKED_INTEGERS

> This table format provides the same information as the unpacked ALGOLW tables, but without the ALGOLW syntax. The values of the functions are punched with a PL/1 format of COLUMN(1), 20 F(4), with the first three functions on the first card, and with each function beginning a new card thereafter. The values of <u>trans</u> are punched starting a new card for each state.

UNPACKED_XPL or UNPACKED_EXPL

> This kind of table format contains the same informa-tion again, but with XPL literals and arrays. NUM_PRODUCTIONS, NUM_STATES, and NUM_SYMBOLS are all literals. A fourth constant, NUM_TRANSITIONS, is also declared literally to be the product of the number of states and the number of symbols. LHS_-

SYMBOL and RHS_LENGTH are arrays whose subscripts are production numbers. TRANS is an array whose elements are a linearized version on the values of <u>trans</u>:

   <u>trans</u>(s,v) = TRANS ((s-1)*NUM_SYMBOLS + v)

The zero'th element of each of the XPL arrays is arbitrarily assigned the value zero.

ALGOLW

This form is a set of ALGOLW function procedures that yield packed parser tables. NUM_PRODUCTIONS, NUM_STATES, LHS_SYMBOL, and RHS_LENGTH are identical to the same procedures in the unpacked ALGOLW tables. NUM_SYMBOLS is replaced with NUM_TRANSI-TIONS, a parameterless function that returns the total of the lengths of all the symbol (or action) lists. The function DEFAULT_ACTION takes a state number and returns the default action for the state. The symbol and action lists are compacted into three functions, FIRST_TRANS, TRANS_SYMBOL, and TRANS_-ACTION. The symbol list for state "s" is the list of values

   TRANS_SYMBOL (FIRST_TRANS(s)),
   TRANS_SYMBOL (FIRST_TRANS(s) + 1),
        ...
   TRANS_SYMBOL (FIRST_TRANS(s+1) - 1)

and the action list for the same state is the list of values

   TRANS_ACTION (FIRST_TRANS(s)),
   TRANS_ACTION (FIRST_TRANS(s) + 1),
        ...
   TRANS_ACTION (FIRST_TRANS(s+1) - 1)

INTEGERS

This form of parser tables provides the same information as the packed ALGOLW tables, but without the ALGOLW syntax. Like the unpacked integer tables, the values of the functions are punched with a PL/1 format of COLUMN(1), 20 F(4), with the first three functions on the first card, and with each function beginning a new card thereafter.

XPL or EXPL

This format is analogous to the packed ALGOLW tables except that XPL arrays are punched in place of the

ALGOLW functions with parameters. NUM_PRODUCTIONS, NUM_STATES, and NUM_TRANSITIONS are punched as literals, and a new, self-explanatory literal is included, NUM_STATES_PLUS_ONE. The zero'th element of each of the XPL arrays is arbitrarily assigned the value zero.

BCPL

This form is analogous to the packed ALGOLW tables except that BCPL arrays are punched in place of the ALGOLW functions with parameters. NUM_PRODUCTIONS, NUM_STATES, and NUM_TRANSLATIONS are punched as MANIFESTs, and a new self-explanatory MANIFEST called NUM_STATES_PLUS_ONE is included. The zero-th element of each of the BCPL arrays is arbitrarily assigned the value of zero.

VOCABULARY

This option punches an entirely different kind of table from the other punch options. This causes the entire vocabulary to be punched, with the symbols in the order of their assigned numbers. Each symbol is punched on a separate card with no punctuation, starting in column one.

Miscellaneous:

LALRGEN does not detect unreachable productions. If a production is unreachable, it will be listed as part of the grammar and its symbols will be numbered, but items derived from it will be absent from the states of the resulting parser.

The program considers lowercase and uppercase letters to be different characters. This can be an advantage, because it allows grammar symbols to be input and listed in a form very similar to the way they often appear in the literature. On the other hand, if such output is mistakenly directed to a PN (uppercase only) printer, the output will be entirely uppercase and potentially very confusing.

The punched parser tables for a nontrivial grammar can be very large, particularly the unpacked tables. There is no guarantee that these tables will satisfy compiler restrictions such as those on the number of constants, number of procedures, or size of arrays.

*LATEX

Contents: The LaTeX text-processing program.

Use: LaTeX is invoked by the $RUN command.

Program Key: *LATEX

Logical I/O Units Referenced:
        SERCOM - error messages, error recovery prompting, and
               diagnostics.
        GUSER - error recovery prompt input.

        LaTeX references all files explicitly by name.

Return Codes: Always zero.

Description: The complete description of the LaTeX text-processing
                language is given in the following publications:

        LaTeX: A Document Preparation System by Leslie
           Lamport,   Addison-Wesley   Publishing   Company
           (1984), ISBN 0-201-15790-X.
        Computing Center Memo 815, "Using LaTeX on MTS."

                LaTeX is a special version of Donald Knuth's TeX text-
                processing system. LaTeX is actually a set of macros for
                use with TeX that simplify the use of TeX dramatically
                and provide a structured document format that handles all
                the typographical details for the user, while providing a
                variety of useful document styles and features not
                available in regular TeX. These include section number-
                ing, cross-referencing, automatic table of contents gen-
                eration, and indexing. The typical output device for
                LaTeX on MTS is the Xerox 9700 page printer.

                Mathematics and technical applications that use special
                characters or accents are LaTeX's strong point; other
                text processors may be more appropriate for simple
                documents. LaTeX is device-independent, which means that
                commands used to produce a document on one device need
                not be changed to produce the same document on another
                device.

                LaTeX accepts an input file containing text to be
                formatted and control codes to direct the formatting
                process and produces a device-independent (DVI) output
                file.   The DVI file is written in internal (binary)
                format. This file must be processed by another program
                to produce readable hard-copy results.   The program

*DVIXER transforms the DVI file into one that can be sent to the Xerox 9700 and printed by running *PAGEPR as the final step.   Support for use of LaTeX with a phototype-setter is available also within certain font limitations.

LaTeX is designed principally for the production of multi-page documents. Although LaTeX is quite capable of producing small documents, the cost associated with initialization of LaTeX may be more than the cost of formatting a small document.

256.2 *LATEX

January 1987

*LCS

Contents: This file contains an LCS loader input record referring to the system low-core symbol dictionary LCSYMBOL. This card placed at the end of the user's object module can reduce the loading time of the program if no library subroutines are referenced.

Use: The LCS input record may be either copied to the end of the user's object module

      $COPY *LCS OBJECT(LAST+1)

or concatenated to the object file on the $RUN command

      $RUN OBJECT+*LCS

Program Key: *EXEC

Description: Normally, the dynamic loader will search the system library *LIBRARY for unresolved symbols. The file *LI-BRARY has an LCS loader input record at the end of it which causes the search of the low-core symbol dictionary if the symbol is not found in *LIBRARY. By appending an LCS input record to the end of the user's object module, the search of the low-core symbol dictionary is be made before the search of *LIBRARY is made. Thus, if the user's object module does not refer to any *LIBRARY symbols, this file will not be searched if all unresolved symbols are found in the low-core symbol dictionary. This will reduce the loading time for the program.

For further information on the dynamic loader and loader records, see the section "The Dynamic Loader" in MTS Volume 5, System Services.

January 1987

<u>*LIBRARY</u>

Contents: The standard MTS subroutine library.

Use: In standard use, *LIBRARY is never explicitly invoked.
                 Instead, after loading the user's explicitly specified
                 object modules, the system loader will implicitly scan
                 *LIBRARY if there remain any unresolved external symbols.
                 This implicit scanning of libraries by the system loader
                 can be controlled by the MTS $SET command options

                     $SET *LIBRARY={ON|OFF}
                     $SET LIBR={ON|OFF}
                     $SET LIBSRCH={ON|OFF}

                 as described in MTS Volume 1, <u>The Michigan Terminal
                 System</u>.

Program Key: *EXEC

Example: If the user specifies

                     $RUN -LOAD+*PLOTSYS

                 then the loader goes through the following steps:

                 (1) The object modules in the file -LOAD are loaded
                      and linked together.
                 (2) Next, the object modules are selectively loaded
                      from *PLOTSYS (since it is a library) to resolve
                      external symbols (i.e., subroutine names) in
                      -LOAD.
                 (3) Finally, if there are still unresolved external
                      symbols, *LIBRARY is searched for the appropriate
                      object modules.

Description: A list of and descriptions of the subroutines contained
                 in *LIBRARY is contained in MTS Volume 3, <u>System Sub-
                 routine Descriptions</u>, along with a description of the
                 format of a library. Note that *LIBRARY is only one of
                 several libraries available in MTS. Others (which must
                 be specified explicitly on a $RUN or $LOAD command) are
                 *CBELLLIB, *CSMPLIB, *EXPLLIB, *ICONLIB, *PASCALJBLIB,
                 *PASCALVSLIB, *PLOTSYS, *PL1FLIB, *PL1OPTLIB, *SPITLIB,
                 *TANGOLIB, and *WATLIB.

January 1987

*LINKEDIT

Contents: The MTS linkage editor.

Purpose: To link-edit files that contain programs in object module
                form.

Use: The program is invoked by the $RUN command.

Program Key: *LINKEDIT

Logical I/O Units Referenced:
                SCARDS - either the input file containing the object
                         modules to be edited, or a sequence of linkage
                         editor commands.
                SPRINT - printed output produced by the linkage editor.
                SPUNCH - default output unit for object modules punched
                         by the linkage editor.
                SERCOM - linkage editor diagnostics and prompting
                         messages.
                GUSER - user responses to prompting (if terminal mode).

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command.   The parameters must be
                separated by commas or blanks.   The minimum acceptable
                abbreviation for each parameter is underlined.

                    COMSAVE/NOCOMSAVE

                        COMSAVE specifies that COM (comment) records are
                        to be saved during linkedit processing.   NOCOM-
                        SAVE specifies that COM records are not saved.
                        The default is NOCOMSAVE.

                    GENSAVE/NOGENSAVE

                        GENSAVE specifies that object module generation
                        information on COM and END records is to be
                        preserved if a combine operation is performed.
                        NOGENSAVE specifies that the generation informa-
                        tion is to be discarded.   The   default   is
                        NOGENSAVE.

                    SYMSAVE/NOSYMSAVE

                        SYMSAVE specifies   that SYM (symbol) records are
                        to be saved during linkedit processing (except
                        during a combine operation). NOSYMSAVE specifies
                        that SYM records are not saved. The default is
                        SYMSAVE.

EMPTY/NOEMPTY

> EMPTY specifies that the output file assigned to SPUNCH is to be emptied before use. NOEMPTY specifies that the output file is not to be emptied. The default is NOEMPTY.

ORL=n

> "n" specifies the maximum output record length to be used for output produced by the linkage editor. The default is the maximum record length of the output file or device.

GAPSIZE=n

> "n" specifies the maximum object module text gap size to be filled during linkage editor processing.

MISCSAVE/NOMISCSAVE

> MISCSAVE specifies that miscellaneous loader records (ALI, DEF, LCS, NCA, OPT, and RIP) are to be preserved during linkedit processing. NOMISC-SAVE specifies that these records are to be deleted. The default is MISCSAVE.

MSGSAVE/NOMSGSAVE

> MSGSAVE specifies that MSG (message) records are to be saved during linkedit processing. NOMSG-SAVE specifies that MSG records are not saved. The default is MSGSAVE.

WXTOER/NOWXTOER

> WXTOER specifies that all ESD (external symbol dictionary) symbols of type WX (weak external reference) are to be converted to type ER (external reference) during linkedit processing. NOWX-TOER specifies that type WX symbols are not converted. The default is NOWXTOER.

TERSE/VERBOSE

> The TERSE/VERBOSE parameters control the amount of information produced by the verification of some commands. TERSE specifies that minimal information is requested; VERBOSE specifies that full information is desired. The default is VERBOSE. This parameter pair has no effect if NOVERIFY is specified.

January 1987

VERIFY/NOVERIFY

VERIFY specifies that verification for each com-
mand is requested. NOVERIFY suppresses the veri-
fication. The default is VERIFY.

Return Codes: None.

Description: The linkage editor in MTS provides a wide range of
facilities for manipulating programs which are in object
module form. These facilities fall roughly into three
categories: making physical changes to object programs,
making logical changes to object programs, and examining
the structure of object programs.

Physical changes to an object program have no effect on
its execution. A large number of physical formats can
represent the same object program. The linkage editor
provides facilities for changing from one physical format
to another so that:

(1) an object program will fit on an alternative
     storage medium (e.g., from a sequential file to
     the card punch),
(2) the CPU time required to load an object program
     is minimized,
(3) the file space required to store an object
     program is minimized.

The linkage editor also provides facilities for changing
the logical structure of object programs. Symbols in
object programs can be replaced or deleted.   Object
modules can be combined, deleted, or replaced. The
loader-defined entry point can be changed.

Interrogative facilities are also available for examining
the logical structure of object programs. Maps and
cross-reference tables of the symbols defined and
referenced in an object program can be displayed.

Those users who want only to reformat object modules can
simply do the following:

     $RUN *LINKEDIT SCARDS=inFDname SPUNCH=outFDname

In this case, object modules will be read from "inFD-
name", converted to linkage editor internal representa-
tion, and written out on "outFDname".   The logical
structure of the modules will be completely and identi-
cally preserved, with the following exceptions:

(1) REP records encountered in input modules will be
     absorbed into the text of output modules.   Input

modules are object modules read in by the linkage
editor.   Output modules are object modules writ-
ten out by the linkage editor.

(2) If multiple ENT records are encountered, all but
the first ENT record encountered are discarded.

(3) Nonabsolute DEF records will be incorporated into
the external symbol dictionary of the appropriate
output modules.

(4) The first LDT record encountered terminates in-
put, but is preserved in the output module(s).

(5) Library control records are ignored. If   a
library is read in, all of its modules will be
written out, minus any library control records.

(6) COM records are discarded.

Those users who wish to use the more advanced features of
the linkage editor must use the linkage editor command
language described in Volume 5, System Services. Com-
mands are read from SCARDS and printed output is written
on SPRINT.   The typical $RUN command to use the linkage
editor in command mode is:

     $RUN *LINKEDIT

Command input is terminated by an end-of-file or by the
STOP command.

Example: $RUN *LINKEDIT SCARDS=OBJ1 SPUNCH=OBJ2

               The object modules in the file OBJ1 are processed by
               the linkage editor and written into the file OBJ2.

January 1987

<u>*LINK11</u>

Contents: The PDP-11 link editor.

Use: The program is invoked by the $RUN command.

Program Key: *LINK11

Logical I/O Units Referenced:
               GUSER - command line input.
               SPRINT - response to the commands.

Return Codes: Always zero.

Description: The MTS loader records produced by one or more runs of
               *11ASR are processed and an absolute binary loader tape
               is produced. The reading of data and writing of the
               results is under the control of the command language.
               The full description of the commands is given in Comput-
               ing Center Memo 286.

Example: $RUN *LINK11

                   The above command invokes the PDP-11 link editor.

January 1987

<div align="center">

*LOCALPLOT
</div>

Contents: A program to plot output on a plotter local to a
                terminal/microcomputer cluster.

Use: The program is invoked by the $RUN command.

Program Key: *LOCALPLOT

Logical I/O Units Referenced:
                SCARDS - file to be plotted.
                SPRINT - program nonplot output.
                SERCOM - program error messages.

Parameters: The following parameters may be specified on the $RUN
                command. These will control the different options for
                the plotting and the plot-file conversion. The minimum
                abbreviation for each parameter is underlined.

                LOCATION=plotter-location

                        The LOCATION option specifies the location of the
                        local plotter. Normally this parameter is not
                        needed since, by default, the plotter will be
                        selected from a system table giving the locations of
                        the plotters for each terminal cluster. This param-
                        eter is only required if the terminal is not in the
                        system table, the terminal is a dial-up terminal, or
                        a different plotter site is desired.

                PLOTTER=plotter-type

                        The PLOTTER option specifies the type of plotter to
                        be used at a certain location. Normally this
                        parameter is not needed, since the program will
                        select the default plotter to use at a certain
                        location. This parameter is only required if there
                        is more than one type of plotter at a single
                        location, and the default type is not desired.

                NOCHECK

                        The NOCHECK parameter specifies that the validity of
                        the input file is not to be checked. The program
                        will normally check the input file to see if it is a
                        valid file for the plotter to use. It is very
                        difficult to completely check a file full of HPGL
                        commands, hence the program will only check the
                        first three characters of that file for the HPGL
                        plotter-on command, which is "ESC . (" or "ESC . Y".

The Hewlett-Packard 7550 Plotter will work properly
without the plotter-on and plotter-off commands, but
every HPGL file generated in MTS will have these as
the first three characters. Files with HPGL com-
mands produced from other sources might not have
these commands at the front. Caution should be used
when using this parameter, since if an incorrect
file is sent to the plotter, unpredictable results
may occur.

SLOW

The SLOW option plots the plot-file at a slower
speed than normal. This will help improve the
quality of the lines drawn, but will also increase
the amount of time taken to draw the plot.

The following parameters only apply to the plot-file
conversions:

CENTER

The CENTER option specifies that the plot is to be
centered on the page. The plot will be positioned
such that its midpoint is mapped to the page
midpoint, i.e., the point $((xmax-xmin)/2)$ of the
plot is mapped to the center of the page (this is
$(5.5,4.25)$ in landscape mode and $(4.25,5.5)$ in
portrait mode). The CENTER option will override any
SHIFT or ORIGIN option specifications. CENTER is
useful only if NOFIT is specified; the plot is
automatically centered in FIT mode.

FIT

The FIT option optimally fits each plot onto the
plotter paper. The Hewlett-Packard 7550 Plotter
uses 8.5 x 11 inch paper with a drawing area on that
paper of 7.75 x 10.0 inches. A fitted plot will be
scaled so that it is as large as possible without
any part of the plot being clipped. FIT is the
default mode.

FRAME={n|(n,m)|ALL}

The FRAME option specifies which plots in the file
are to be drawn. For example, FRAME=5 will draw the
fifth plot, FRAME=(3,6) will draw the third through
sixth plots, and FRAME=ALL will draw all the plots.
The default is FRAME=ALL.

LANDSCAPE

The LANDSCAPE option specifies that the plot drawn is to be oriented in landscape mode, i.e., the x-axis will be parallel to the 11-inch edge of the paper.

NOFIT

The NOFIT option specifies that the coordinates to be used in each plot are in absolute inches. No implicit scaling is performed on the plots, and coordinates which are outside of the drawing area on the plotter paper (10.0 inches for "x" and 7.75 inches for "y" on the HP 7550) will be clipped appropriately. The origin point (0.0,0.0) is at the extreme bottom left-hand corner of a page, which means that coordinates in the negative axis range will be clipped.

ORIGIN=(x,y)

The ORIGIN option may only be used in conjunction with the NOFIT option. By default, the origin point (0.0,0.0) of a NOFIT plot is located at the extreme bottom left-hand corner of a page. The ORIGIN option allows the location (0.0,0.0) to be relocated to an arbitrary point, not necessarily within the boundaries of a physical page. It is up to the user to ensure that a plot remains within the visible region if this option is used.

Either the "x" value, the "y" value, or both, may be omitted from an ORIGIN command; unspecified parameters will default to 0.0. A maximum of three digits following the decimal point may be given.

The ORIGIN specification is not affected by the presence of a SCALE specification (i.e., the origin coordinates are not scaled). Using the effects of these two options, it is possible to increase the visible viewing region of a plot.

PEN(color,size)=newcolor

The PEN option respecifies the pen colors. "color" and "size" specify the color and size of the pen to be changed; "newcolor" specifies the new color of the pen. The "size" parameter may be omitted if the "color" parameter unambiguously specifies the pen to be changed, e.g., there is not both a medium and fine pen to the same color. For example, to have the only red pen in the plot drawn as blue, specify

PEN(RED,)=BLUE; to have a fine black pen drawn as
green, specify PEN(BLAC,FINE)=GREE. If a PPEN re-
cord in a plotfile specifies a nonsupported color,
that color will be drawn as black.

PORTRAIT

The PORTRAIT option specifies that the plot drawn is
to be oriented in portrait mode, i.e., the x-axis
will be parallel to the 8.5-inch edge of the paper.

SCALE=scale-factor

The SCALE option specifies a scaling factor which
must be in the range 0.0 < scale-factor ≤ 1.0. In
FIT mode, this scaling is applied to the image after
it has been fitted onto a page. The default scale
factor is 1.0.

SHIFT

The SHIFT option may only be used in conjunction
with the NOFIT option. By default, a plot produced
in NOFIT mode has its origin at the bottom left-hand
corner of a page. Specifying the SHIFT option will
cause each image to be translated up by one inch to
give more space at the bottom of the page. The
SHIFT option is the same as specifying
ORIGIN=(0.0,1.0).

Return Codes: 0 - Successful return.
              4 - Unable to generate plot.

Description: This program is designed to allow users working at a
             terminal/microcomputer cluster plot output on a plotter
             at that location. The program can either take a file
             containing commands for the plotter, or an MTS plot-file,
             which will be automatically converted. The program will
             automatically select the appropriate UMnet network
             address for the local plotter from a system table, and
             then route the output and necessary plotter controls to
             that plotter.

             See Computing Center Memo 465, "*LOCALPLOT: Plotting at
             Remote Stations," for further information on plotting
             files on local plotters.

Example: $RUN *LOCALPLOT SCARDS=PLOTFILE

             In the above example, the plot contained in the file
             PLOTFILE is plotted on the local plotter.

*LOCALPRINT

Contents: A program to print output on a printer local to a
               workstation cluster.

Use: The program is invoked by the $RUN command.

Program Key: *LOCALPRINT

Logical I/O Units Referenced:
               SCARDS - output to be printed.
               SPRINT - program error messages.

Parameters: The following parameters may be specified on the $RUN
               command. The following parameters apply to all types of
               local printers.

               LOC=xxx   The LOC parameter specifies the location of
                            the local printer. Normally this parameter
                            is not needed since, by default, the printer
                            will be selected from a system table giving
                            the locations of the printers for each
                            terminal cluster. This parameter is only
                            required if the terminal is not in the
                            system table, the terminal is a dial-up
                            terminal, or the user wishes to select a
                            different printer site.

               BINARY   The BINARY parameter specifies that the
                            output is to be printed without any conver-
                            sion. Normally, the program converts the
                            output to ASCII and adds printer controls
                            that are appropriate to the local printer.
                            The BINARY parameter is normally only used
                            when printing output from certain text pro-
                            cessors (e.g., T^3 or Microsoft WORD) which
                            have their own printer driver programs for
                            the local printer.

               NOCC   The NOCC parameter specifies that the output
                            file does not contain carriage-control
                            characters.

             | PRETDAY   PRETDAY assumes that the output file con-
             | tains pre-T-Day translation codes (i.e., the
             | file was created before February 22, 1988
             | and not translated). See Computing Center
             | Memo 480, "Translation Day Character Code
             | Changes on MTS."

The following parameters apply only to the Xerox 2700 or
the Xerox 4045 local page printers.

PORTRAIT or   This parameter specifies either portrait or
LANDSCAPE   landscape orientation for the output.   The
              default is LANDSCAPE.

MARGIN=xx.x   The MARGIN parameter specifies the left
              margin width (in inches). By default, the
              left margin is .667 inches for landscape and
              0.5 inches for portrait output.

RESET   The RESET parameter sends a reset sequence
              to the printer before the job is started.
              Normally, this is not required except when
              the previous job was fatally terminated
              while printing output. If the current job
              is printing in unexpected typefaces, then
              rerunning *LOCALPRINT with RESET specified
              may correct this problem.

| HEAD/NOHEAD   The HEAD parameter specifies that an identi-
              fying headsheet is to be printed with each
              job.   The NOHEAD parameter specifies that
              the headsheet is to be omitted. The default
              is HEAD for the Xerox 4045 printer and
              NOHEAD for the Xerox 2700 printer.

NOREV   The NOREV parameter specifies that the pages
              are not to be printed in the reverse order.
              This parameter applies only to the Xerox
              4045 printer.

Return Codes: 0 - Successful return.
               4 - Empty input file or stopped by attention interrupt.
               8 - Error termination (no output printed).

Description: This program is designed to allow users working at a
              workstation cluster to print output on a local printer
              attached to the cluster.   The program automatically
              selects the appropriate network address for the local
              printer from a system table and then routes the output
              and necessary printer controls to that printer.

              For the Xerox 2700 and 4045 page printers, an identifying
              headsheet is printed with the output so that output from
              one job can be distinguished from the next job.   This
              headsheet may be suppressed by specifying the NOHEAD
              parameter.

              For the Xerox 4045 page printer, the pages are printed in
              the reverse order so that they will be in the correct

order when they are stacked. This may be overridden by
specifying the NOREV parameter.

Example: $RUN *LOCALPRINT SCARDS=MYOUTPUT

                In the above example, the output contained in the
                file MYOUTPUT is printed on the local printer.

272.2 *LOCALPRINT

<u>*LOGPURGE</u>

Contents: A program to delete lines from a file based on a date
                recorded in each line (a log file) or from a file
                produced by the MTS $LOG command.

Use: The program is invoked by the $RUN command.

Program Key: *LOGPURGE

Logical I/O Units Referenced:
                0   - the name of the line file to be purged.
                SERCOM - error comments.

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command.   The parameters must be
                separated by blanks or commas; the underlined portion of
                each parameter may be used as an abbreviation.

                <u>RETENTION</u>   This parameter specifies the number of days,
                            weeks, months, or years that a line is to be
                            retained in the file.   This parameter is
                            specified in the form

                                [RETENTION=]n[{D|W|M|Y}]

                            The default is 7 days (7D). A month (M) is
                            defined to be 365/12 days, and a year is
                            defined to be 365 days.

                <u>DATEOFFSET</u> This parameter specifies the offset from the
                            beginning of each line in the file for the
                            date used to determine if the line is to be
                            purged. The parameter is specified in the
                            form

                                DATEOFFSET=n

                            The default is zero.   This parameter is
                            ignored for UNIT=$LOG.

                <u>R</u>ENUMBER   This parameter specifies that the file is to
                            be renumbered after purging starting with
                            line 1 and incrementing by 1.

                <u>U</u>NIT   This parameter specifies the unit for the
                            date. The parameter is specified in the form

                                UNIT={<u>CHAR</u>|<u>DAYS</u>|<u>M</u>INUTES|<u>S</u>ECONDS|<u>MS</u>ECONDS|
                                     <u>A</u>LPHA|$LOG}

                        The default is CHAR (mm/dd/yy). CHAR is 8
                        bytes in length; MSECONDS (microseconds) is
                        an 8-byte, binary integer; ALPHA (mmm dd/yy)
                        is a 9-byte, character representation of the
                        date using 3-character abbreviations for the
                        months; $LOG is produced by the MTS $LOG
                        command; the other units are 4-byte, binary
                        integers. DAYS, MINUTES, SECONDS, and MSE-
                        CONDS are based on March 1, 1900 with 0
                        equivalent to 03/01/00 00:00:00.000000.

| Return Codes: 0 - Successful return
| 4 - Error return

  Description: Each line of the file assigned to logical I/O unit 0 is
                 deleted until a line is encountered which has a date more
                 recent than the current date minus the number of days
                 specified by the RETENTION parameter.

  Example: $RUN *LOGPURGE 0=ONEWEEK PAR=RENUMBER

                        In the above example, all records in the file
                        ONEWEEK are deleted until the date, in the form
                        mm/dd/yy starting in column position 1, is greater
                        than or equal to the date of the day 7 days before
                        the current day. The lines in the file are then
                        renumbered.

                 $RUN *LOGPURGE 0=TWOMONTHLOG PAR=2M D=1 R

                        In the above example, all records in the file
                        TWOMONTHLOG are deleted until a record is encoun-
                        tered with a date greater than or equal to the date
                        of the day 61 days before the current day. The date
                        is 8 characters starting in column position 2 of the
                        record. After deletion, the remaining lines are
                        renumbered.

                 $RUN *LOGPURGE 0=LOG PAR=RET=3 R U=$LOG

                        In the above example, all records in the file LOG
                        are deleted until a record is encountered beginning
                        with "1$Log output:" and ending with a date of the
                        form "mmm dd/yy" that is greater than or equal to
                        the date of the day 3 days before the current day.
                        The file is produced by the MTS $LOG command. After
                        deletion, the remaining lines are renumbered.

January 1987

<u>*MACBINHEX</u>

Contents: A program to convert Macintosh files from Binhex 4.0
                format to the Macintosh External File System (EFS)
                format.

Use: The program is invoked by the $RUN command.

Program Key: *MACBINHEX

Logical I/O Units Referenced:
                SCARDS - the input file containing the Binhex 4.0 file to
                         be converted.
                SPUNCH - the output file to receive the converted file.

Return Codes: 0 - Successful return.
                4 - Invalid Binhex file.
                8 - Checksum error. File corrupted.

Description: The *MACBINHEX program takes a Binhex 4.0 formatted file
                that has been uploaded to MTS from a Macintosh microcom-
                puter and converts it to a file which can be used with
                the Macintosh External File System (EFS), *MACPAINT4045,
                or *MACWRITE9700.   For information on uploading MTS
                files, see Computing Center Memo 904, "Using the Macin-
                tosh as a UMnet terminal: The Kermit Program."   For
                further information on Macintosh conversion programs, see
                Computing Center Memo 905, "Macintosh to MTS Conversion
                Programs."

Example: $RUN *MACBINHEX SCARDS=BINHEXFILE SPUNCH=MACEFSFILE

January 1987

<u>*MACPAINT4045</u>

Contents: A conversion program allowing MacPaint files to be
                 printed on a Xerox 4045 printer.

Use: The program is invoked by the $RUN command.

Program Key: *MACPAINT4045

Logical I/O Units Referenced:
                 SCARDS - input file containing a MacPaint picture.
                 SPUNCH - output file ready for *LOCALPRINT.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command. The case and positioning of
                 the parameters is unimportant. If a parameter is speci-
                 fied more than once, the right-most value will be used.
                 The underlined portion is the minimum abbreviation for
                 that option.

                 {MAGNIFICATIONFACTOR|MAGFAC}={1|2|4} Default: 4

                     The MAGNIFICATIONFACTOR determines the size of the
                     processed image on the page. MAGNIFICATIONFACTOR=4
                     is the default, which indicates that the MacPaint
                     picture should be expanded to fill the entire page.
                     MAGNIFICATIONFACTOR=2 produces a picture one quarter
                     (1/4) the size of the page, and MAGNIFICATIONFACTOR=
                     1 produces a picture one-sixteenth (1/16) the size
                     of the page.

                 <u>CENTERED</u> / <u>NOCENTERED</u>   Default: NOCENTERED

                     The CENTERED option centers the printed image on the
                     page.   The option is ignored if POSITION=(x,y) is
                     specified in the PAR field.

                 BORDER / NOBORDER   Default: NOBORDER

                     The BORDER option puts a small blank border around
                     the printed picture. This option is valid only if
                     the POSITION=(x,y) option is used, otherwise it is
                     ignored.

                 <u>POS</u>ITION=(x,y)

                     The POSITION=(x,y) option allows specifies where on
                     the page the picture is placed. "x" and "y" can
                     have values from 1 to 4. The "x" axis is along the
                     short edge of the page and the "y" axis is along the

long edge of the page. POSITION=(1,1) indicates the
upper-left corner of the page (in portrait orienta-
tion) while POSITION=(4,4) indicates the lower-right
portion of the page. If the x-y coordinate is one
of (1,1), (1,2), (2,1), or (2,2), the MAGNIFICATION-
FACTOR will default to 2 and the picture will cover
one quarter of the sheet. Positions (1,3), (1,4),
(2,3), (2,4), (3,1), (3,2), (3,3), (3,4), (4,1),
(4,2), (4,3), (4,4) default to MAGNIFICATIONFACTOR=1
and cannot have any other MAGNIFICATIONFACTOR. A
MAGNIFICATIONFACTOR larger than   this   will   be
ignored, but a smaller MAGNIFICATIONFACTOR will be
used if specified.

Return Codes: 0 - Successful return.
               4 - Invalid MacPaint file.

Description: The program *MACPAINT4045 takes a MacPaint file, which
must first be transferred from the Macintosh to MTS, as
input and prepares an output file containing the pro-
cessed MacPaint file, which can then be printed on the
Xerox 4045 printer in portrait orientation. For informa-
tion on uploading MacPaint pictures from the Macintosh to
MTS, see Computing Center Memo 904, "Using the Macintosh
as a UMnet Terminal: The Kermit Program." To produce a
picture ready for the Xerox 4045, issue the command:

      $RUN *MACPAINT4045 SCARDS=paintfile SPUNCH=output
          [PAR=options]

where "paintfile" is the MacPaint file which has been
uploaded to MTS and "output" is a file to hold the
processed MacPaint picture. To finally print the pic-
ture, issue the command:

      $RUN *LOCALPRINT SCARDS=output PAR=BINARY

For further information on Macintosh conversion programs,
see Computing Center Memo 905, "Macintosh to MTS Conver-
sion Programs."

Example: $RUN *MACPAINT4045 SCARDS=PAINTFILE SPUNCH=XEROXFILE

In the above example, the MacPaint picture in the
file PAINTFILE is converted to a print file for the
Xerox 4045 printer.

January 1987

*MACUTIL

Contents: The MTS macro-library editor.

Purpose: To edit files that contain 370 assembler language or
                 *FORMAT macros.

Use: The program is invoked by the $RUN command.

Program Key: *MACUTIL

Logical I/O Units Referenced:
                 SCARDS - the input file containing the macros to be
                          replaced.
                 SPRINT - printed output produced by the macro-library
                          editor.
                 SERCOM - diagnostic messages.
                 GUSER - a sequence of commands or user responses in
                          conversational mode.
                 0   - default unit for the macro library to be edited.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command. The parameters must be
                 separated by commas or blanks. Parameters may be negated
                 by "-", "˜", "NO", or "N". The minimum acceptable
                 abbreviation for each parameter is underlined.

                     BREAK=n

                         Each new macro or copy section added to the edit
                         file begins at the next highest multiple of the
                         line number "n" after the last line in the macro
                         library. Lines currently in the edit file are
                         not changed to reflect the new value. The
                         default value is 100.000. No fractions are
                         permitted.

                     BUILDIR

                         The BUILDIR parameter specifies that the macro-
                         library editor is to produce a directory in the
                         macro library. The library is assumed to have no
                         directory but just a series of macros and/or copy
                         sections. The default is NOBUILDIR; i.e., the
                         macro library should not have the directory
                         generated.

EMPTY

The EMPTY parameter specifies that the macro library assigned to unit 0 is to be emptied before use. NOEMPTY specifies that the macro library is not to be emptied. The default is NOEMPTY. EMPTY has no effect in command mode.

HDRGEN

The HDRGEN parameter specifies that MACUTIL generate a copy-section header whenever a copy section is added with the NAME modifier in the ADD, INCLUDE, REPLACE, and UPDATE commands. NO-HDRGEN specifies that the copy-section header should not be generated. The default is HDRGEN.

INCREMENT=n

The INCREMENT parameter may be used to increment each successive line in the macro library. The default is 1.000.

QUIT

The QUIT parameter specifies that if the macro-library editor encounters any errors in batch mode, the user is signed off. NOQUIT specifies that the batch user is not to be signed off if an error occurs. The default is NOQUIT.

SEQ

The sequence numbers in columns 73-80 are normal-ly not removed. But if SEQ=OFF, they are deleted and the SEQ parameter can be used to truncate any new macros. The default is SEQ=ON.

START=n

The START parameter specifies that, if a macro library is empty or is to be renumbered, the first line of a first macro should start at MTS line number n. The parameter should be an integral number, and the default is 1000.

TERSE/VERBOSE

The TERSE/VERBOSE parameters control the amount of information produced by the verification of some commands. TERSE specifies that minimal information is requested; VERBOSE specifies that full information is desired. TERSE is an antony-

mfor VERBOSE. The default is the setting of the
MTS TERSE option.   This parameter pair has no
effect if NOVERIFY is specified.

UPDATE/REPLACE

The UPDATE/REPLACE parameters control whether the
file assigned to SCARDS is an update or replace-
ment file in no-command mode. UPDATE specifies
that macros or copy sections read from SCARDS not
in the edit file will be included. REPLACE
specifies that new macros will be excluded. The
default is UPDATE. The parameters are effective
only if SCARDS is specified in the $RUN command.

VERIFY

The VERIFY parameter specifies that verification
for each command is requested. NOVERIFY sup-
presses the verification. The default is VERIFY.

Return Codes: 0 - Successful return.
               4 - Errors detected.
               8 - Errors detected.
               12 - Unsuccessful return.

Description: The MTS macro-library editor is used for the editing of
               macro libraries. Many *OBJUTIL facilities, such as
               replacing macros, are incorporated in this macro-library
               editor.   The macro-library editor provides facilities to
               replace, add, delete, or correct a macro.

               Those users who want only to update their macro library
               can simply issue the following command:

                   $RUN *MACUTIL SCARDS=inFDname 0=editFDname

               In the above case, macros are read from "inFDname", and
               written on "editFDname" replacing any previous macros.
               The logical structure of the macros is completely pre-
               served except that the duplicate macros from "inFDname"
               are discarded.   Unless the parameter REPLACE is speci-
               fied, additional macros are also added to the macro
               library.

               A typical example is a file consisting of several macros
               in the file MAC. While assembling the macros, the user
               discovers an error in one of the macros. This error may
               be corrected in the source file by the MTS file editor.
               After the macro is checked, the macro-library editor may
               be used to replace the erroneous macro in the macro
               library as follows:

```
$RUN *MACUTIL SCARDS=MAC 0=MACLIB
```

In this example, the source code for the macros replaces
corresponding macros in the file MACLIB.   In addition,
any macro that exists in MAC but not in MACLIB is also
added.

Users who only want to build the directory in the macro
library can simply issue the following command:

```
$RUN *MACUTIL 0=editFDname PAR=BUILDIR
```

The first record of the file "editFDname" should not
start at line 1; otherwise, *MACUTIL will not be able to
build the directory properly (the directory starts at
line 1).   It is recommended that the first record start
at line 1000; this will allow enough room for the
directory.

Those users who desire to use the more advanced features
of the macro-library editor must use the macro-library
editor command language which is described in MTS Volume
14, 360/370 Assemblers in MTS. Commands are read from
GUSER and printed output is written on SPRINT.   The
typical $RUN command to use the macro-library editor in
command mode is

```
$RUN *MACUTIL
```

Command input is terminated by an end-of-file or by the
STOP command.

January 1987

*MACWRITE9700

Contents: A conversion program allowing MacWrite files to be printed on a Xerox 9700 page printer.

Use: The program is invoked by the $RUN command.

Program Key: *MACWRITE9700

Logical I/O Units Referenced:
                  SCARDS - input file containing a MacWrite 4.5 document.
                  SPUNCH - output file ready for the Xerox 9700 page printer.

Return Codes: 0 - Successful return.
                  4 - Invalid MacWrite file.

Description: The program *MACWRITE9700 takes a MacWrite 4.5 file, which must first be transferred from the Macintosh to MTS, as input and prepares an output file containing the processed MacWrite file, which can then be printed on the Xerox 9700 page printer. For information on uploading MacWrite files from the Macintosh to MTS, see Computing Center Memo 904, "Using the Macintosh as a UMnet Terminal: The Kermit Program." For further information on the MACWRITE9700 program, see Computing Center Memo 905, "Macintosh to MTS Conversion Programs."

January 1987

*METATRAN

Contents: The DI-3000 Metafile Translator.

Use: The appropriate DI-3000 library file should be concate-
                 nated to the DI-3000 Metafile Translator on the $RUN
                 command, e.g.,

                     $RUN *METATRAN+*DI3000.TX4010

Program Key: *EXEC

Logical I/O Units Referenced:
                 5   - Input (default: *SOURCE*)
                 6   - Output (default: *SINK*)
                 90   - Segment Storage Output (default: *SINK*)
                 91   - Segment Storage Input (default: *SOURCE*)
                 92   - Metafile System Output (default: *SINK*)
                 93   - Metafile System Input (default: *SOURCE*)
                 94   - Device Driver Output (default: *SINK*)
                 95   - Device Driver Input (default: *SOURCE*)
                 96   - Metafile Translator Alternate Input Unit
                 97   - Metafile Translator Logging (-METALOG(*L+1))
                 99   - Scratch

Return Codes: Always zero.

Description: The DI-3000 Metafile Translator is an interactive program
                 that postprocesses DI-3000 metafiles, sending graphics
                 output to a selected graphics device. A metafile con-
                 tains device independent picture information, each pic-
                 ture being equivalent to the sequence of DI-3000 calls
                 that first defined the picture. Pictures may be read
                 from up to five metafiles concurrently.   They may be
                 positioned, scaled, and superimposed on a selected gra-
                 phics device.   In addition, pictures from different
                 metafiles may be combined and written to a new metafile.

                 For a more complete description of DI-3000 and the
                 DI-3000 Metafile Translator, see Computing Center Memo
                 463, "Using the DI-3000 Graphics Library in MTS."

January 1987

*MPS

Contents: The Mathematical Programming System (MPS).

Purpose: To solve linear programming problems and separable pro-
                 gramming problems.

Use: The program is invoked by the $RUN command.

Program Key: *MPS

Logical I/O Units Referenced:
                 SCARDS - the MPS source program and data. The program
                          and data must be separated by a blank card.
                 SPRINT - the output produced by the program.

Return Codes: 0 - Successful return.
                 8 - Error return.

Description: MPS is an IBM program composed of a set of procedures
                 that permit the solution of linear programming problems
                 and separable programming problems. The full description
                 of MPS is given in the IBM publication, Mathematical
                 Programming System/360, form GH20-0476.

Example: $RUN *MPS
                         PROGRAM('ND')
                         INITIALZ
                         MOVE(XDATA,'EXAMPLE')
                         MOVE(XPBNAME,'PBFILE')
                         CONVERT('SUMMARY')
                         SETUP
                         MOVE(XOBJ,'COST')
                         MOVE(XRHS,'RHS1')
                         PRIMAL
                         SOLUTION
                         EXIT
                         PEND

                 NAME   EXAMPLE
                 ROWS
                  N   COST
                  L   ONE
                  G   TWO
                  E   THREE
                 COLUMNS
                     C1   ONE   3.0   TWO   -2.0
                     C1   THREE   2.0   COST   1.0
                     C2   ONE   2.0   TWO   1.0
                     C2   THREE   2.0   COST   2.0

```
        C3       ONE      3.0      TWO     -2.0
        C3       THREE    1.0      COST     3.0
        C4       ONE     -1.0      TWO      1.0
        C4       THREE    1.0      COST     4.0
    RHS
        RHS1     ONE     25.0      TWO      5.0
        RHS1     THREE   20.0
    ENDATA
    $ENDFILE
```

January 1987

*MTSBIBLIOGRAPHY

Contents: A literature reference list of articles about the design
              of MTS.

Use: The $COPY command should be used, e.g.,

                  $COPY *MTSBIBLIOGRAPHY

Description: The list of articles is organized by author.

January 1987

*MVC

Contents: A record rearrangement program (move characters).

Purpose: Permits the user to rearrange the characters in a record
                 according to a specified pattern, and to insert literal
                 characters and/or sequence fields into each record.

Use: The program is invoked by the $RUN command.

Program Key: *MVC

Logical I/O Units Referenced:
                 SCARDS - input records to be rearranged.
                 SPUNCH - output records after rearrangement.
                 SERCOM - error comments, prompting messages, and summary
                          information.
                 GUSER - user responses to prompting messages (if any).

Return Codes: 0 - Successful return.
                 4 - Control command error (batch mode only).

Description: The user provides a pattern that describes the organiza-
                 tion of output records produced by *MVC.   The pattern
                 consists of one or more elements that control the
                 selection of characters to be placed in an output record
                 and their position in the record.

                 Several types of elements may be included in the speci-
                 fied rearrangement pattern; consecutive elements must be
                 separated by a single comma. The general form of the
                 pattern is

                     e(1),e(2),...,e(i),...,e(n)

                 where e(i) is an element of the pattern.   The program
                 scans the pattern and produces a logically equivalent
                 internal pattern that is applied to all subsequent input
                 records.   An error comment is produced if an error is
                 detected while scanning the elements of the pattern.
                 Scanning of the pattern is terminated when the last
                 character is encountered or when a blank is encountered.

                 The pattern may contain up to 128 elements. Pattern
                 elements are of seven types:

                     (1) Single-Column Element

                          The single-column element is a decimal integer
                          "dd" such that 1 ≤ dd ≤ 32767. This element

specifies that the character in position "dd" of
the input record should be placed in the next
available position of the output record.

(2)   Column-Range Element

The column-range element has the form "bb"-"ee",
where "bb" and "ee" are decimal integers such
that 1 ≤ bb and ee ≤ 32767.   This element
specifies that the characters in positions "bb"
through "ee" (inclusive) of the input record are
to be placed in the next available positions of
the output record. "bb" may be less than, equal
to, or greater than "ee". When "ee" is less than
"bb", field reversal occurs.   When they are
equal, a single column is specified.

(3)   Skip Element

A skip element may take the form "Sdd" or "sdd",
such that 1 ≤ dd ≤ 32767, and specifies that the
next "dd" positions of the output record are to
be skipped over.

(4)   Tab Element

A tab element may take the form "Tdd" or "tdd",
such that 1 ≤ dd ≤ 32767, and specifies that the
next specified character of the output record
should appear in position "dd". Tab elements may
be used to specify any position in the output
record, either before or after the current
position.

(5)   Literal Element

The literal element has the form 'xxxxx' where
"xxxxx" is a string of literal characters that
are to appear in the next available positions of
every output record. The total number of literal
characters may not exceed 256. If a literal is
to contain the character prime, it must be
represented in the literal pattern as two primes.
Thus, the literal pattern element

        'ABC''JKL''''XYZ'

corresponds to the literal string ABC'JKL''XYZ
which occupies the next twelve output positions.
Blanks included in a literal sequence element do
not terminate the pattern scan.   Null literals
are illegal.

Two special literals, D and T (or d and t), are available for the insertion of the date and time into each output record. These literals are presented without primes. The D pattern element is equivalent to 'MM-DD-YY' and T is equivalent to 'HH:MM.SS', where the date or time is obtained immediately prior to the pattern scan. Each use of D or T uses 8 of the 256 available literal character positions.

(6) Decimal Sequence Element

The decimal sequence element has the form "Qnn" or "qnn", where "nn" is an integer such that $1 \leq nn \leq 15$. "nn" represents the width of the sequence field. The sequence numbers start at 1 and are incremented by 1 for each output record.

Modifiers may be appended to the basic sequence-field element to allow the user to specify different initial sequence numbers, different increments, and different moduli for the sequence fields. The form for a sequence element with modifiers is Qnn(bb,ii,mm) where bb, ii, and mm are integers such that $1 \leq bb$, and ii and mm ≤ 999999999999999 (15 decimal positions).

"bb" is the beginning number, "ii" is the increment, and "mm" is the modulus. The default values for "bb" and "ii" are 1 and the default value for the modulus is 10 to the power "nn". If any of the default values are acceptable, they need not be explicitly presented. Thus, if only the beginning number is to be specified, then Qnn(bb) is sufficient. Similarly, if only the increment is to be specified, then Qnn(,ii) is sufficient. The comma indicates that the value presented is the increment, not the beginning number.

Only the "nn" least significant positions of a sequence counter appear in the output record even though the arithmetic operations are carried out to 16 decimal positions. A maximum of 32 decimal sequence elements may be specified in the pattern.

(7) Hexadecimal Sequence Element

This sequence element is similar to the decimal sequence element and has the form "Xnn" or "xnn" where "nn" is a hexadecimal integer such that $1 \leq nn \leq 8$. In the modified form, the beginning,

increment, and modulus values are given in their hexadecimal representation. Either upper- or lowercase letters may be used to represent the hexadecimal digits A...F. The acceptable range of values for these modifiers is 0 through FFFFFFFF inclusive (8 hexadecimal positions). The default values for "bb" and "ii" are 1. The default value for the modulus is sixteen to the power "nn". All arithmetic is carried out in double-precision integer form, but only the "nn" least significant positions appear in the output record. A maximum of 32 hexadecimal sequence elements may be specified in the pattern.

Several sample patterns follow:

S5,1-10,T50,25-21

This pattern specifies that the first five output record characters are to be blanks, that columns 6 through 15 of the output record are to be taken from columns 1 through 10 of the input record, that columns 16 through 49 of the output record are to contain blanks, and that columns 50 through 54 of the output record are to be taken from columns 25 through 21 of the input record.

1-72,'DECK',Q4

This pattern specifies that columns 1 through 72 of the output record are to be obtained from columns 1 through 72 of the input record, columns 73 through 76 of every output record are to contain the word DECK, and columns 77 through 80 of the the output records are to contain sequence numbers starting at 1 and incremented by 1 for each record. The modulus for the sequence numbers is 10000. Thus, the sequence number 9999 is followed by 0000.

1-72,'DECK',Q4(10,5)

This pattern is the same as the previous example except that the initial sequence number is 10 and the increment is 5. The modulus is still 10000, so that the sequence number 9995 is followed by 0000.

1-72,'DECK',Q4(10,5,2000)

This pattern is similar to the previous two examples except that the modulus is 2000. Thus, the sequence number 1995 is followed by 0000.

Input records may range in length from 0 to 32767 bytes;
they are padded with trailing blanks as required. Output
records may range in length from 1 to 32767 bytes (no
null patterns) and trailing blanks are produced if the
pattern specifies them. The trimming of trailing blanks
is under the control of the @TRIM FDname modifier for the
SPUNCH assignment.

For either batch or conversational usage, the PAR field
of the $RUN command may be used to provide the pattern
information. If the pattern does not appear in the PAR
field, the logical I/O unit GUSER is used to read the
pattern.

Since GUSER defaults to the pseudodevice *MSOURCE*, the
batch user may enter the pattern information on the card
following the $RUN command. The conversational user will
be prompted to enter a pattern if there is no PAR field
or if an error is detected in scanning the pattern.

An end-of-file condition on GUSER or SCARDS terminates
execution of the program.

After the pattern has been analyzed and prior to reading
the first record from SCARDS, a message giving the length
of the pattern is printed on SERCOM. After an end-of-
file condition is detected on SCARDS, a message giving
the number of records processed is printed on SERCOM.

Examples: $RUN *MVC SCARDS=FILE1 SPUNCH=FILE2 PAR=1-40,S30,71-80,D

In the above example, the input file is FILE1 and
the output file is FILE2; the pattern is 1-40,S30,
71-80,D. Columns 41-70 of every output record will
contain blanks but the contents of columns 1-40 and
71-80 of the input records will be preserved in the
corresponding columns of the output records. In
addition, columns 81-88 of every output record will
contain the date.

$RUN *MVC SCARDS=FILE1 SPUNCH=FILE2@I PAR=1-72

In this example, all lines of FILE1 will be trun-
cated to 72 characters and written into FILE2. The
line numbers of FILE1 will be preserved in FILE2
since the @I FDname modifier is specified.

January 1987

*NETCOPYMAC

Contents: An MTS command macro library containing macros for
                 copying files across the Merit/UMnet computer network.

Program Key: *EXEC

Logical I/O Units Referenced:
                 None.

Description: When MTS command macros are enabled by the MTS command
                 $SET MACROS=ON, this macro library may be attached to
                 obtain a set of command macros useful for copying files
                 across the Merit/UMnet computer network, e.g.,

                     MACROLIB *NETCOPYMAC

                 The macro library contains two principle macros, NETCOPY
                 and BATCHCOPY. The syntax for NETCOPY is

                     NetCopy FromFile ToFile Destination={Local|Remote}

                 This macro may be used with the NET CLS to copy files
                 between the local and remote host in either direction.
                 It copies "FromFile" to "ToFile" where the direction is
                 given by the "Destination" parameter, which must be
                 either "Local" or "Remote". It must be invoked from
                 within the NET CLS on some MTS machine while it is
                 connected to another MTS machine that is in MTS command
                 mode. Since the NET CLS reads from *SOURCE* with
                 @MACRO@MFR, the NetCopy must be prefixed with a ">"
                 character to have it recognized as a macro call. NETCOPY
                 always copies the entire file, preserves line numbers,
                 and ignores "$CONTINUE WITH" and "$ENDFILE" lines. It
                 also copies the following attributes of the file: Size,
                 Type, Nosave, Prot, Pkey, and Maxsize.

                 The syntax for BATCHCOPY is

                     BatchCopy LclFile RmtFile Destination= Proute=UM
                         T=30 CCID= CKID Rate= Permit=

                 This macro copies files from one MTS merit host to
                 another, by default, from the current host to UB unless
                 the current host is UB in which case it copies to UM.
                 The "remote" host cannot be the same as the "local" one.
                 It creates a network *BATCH* job to sign onto the remote
                 host under the current CCid, copying the files using
                 MNET:BLOCK. "LclFile" may specify a single file, a list
                 of file names in parentheses, or a file pattern. "Rmt-

File" may be specified only if "LclFile" is a single file. If omitted, the local name is used. This macro prompts for the password to be used to signon to the remote host. The other options are:

     CCID   - specify CCid to signon on to different CCid than the current one, i.e., to copy to a different ccid.
     CKID   - specify CKID if CCid is protected by *CKID (to prompt for a second level password).
     DESTINATION - where to copy the files (the XROUTE).
     PROUTE - where to print the batch job listing, UM by default.
     T=30   - time limit used for the batch job (30 seconds).
     RATE   - specify for nondefault rate-group.
     PERMIT - specify to permit file(s) explicitly, e.g., Permit="R P=W?".

For documentation on using MTS command macro libraries, see MTS Volume 21, <u>MTS Command Extensions and Macros</u>.

*NETMAILHELP

Contents: Instructions for sending messages to remote sites not
               listed in *NETMAILSITES.   This file also contains the
               remote mail address to be used by anyone sending a remote
               message to UM or UB.

Use: The $COPY command should be used, e.g.,

                   $COPY *NETMAILHELP

*NETMAILSCHEDULE

Contents: A program to print the schedule of exchange times and the
               completion times of the most recent exchanges of messages
               with remote sites.

Use: The program is invoked by the $RUN command.

Program Key   *NMSCHED

Description: For each remote site that is directly contacted by MTS,
               the program prints the time the last exchange was
               completed and the scheduled times of the next exchanges.

*NETMAILSITES

Contents: A list of the remotes sites to which messages may be
               sent.

| Use: This list has been discontinued.

January 1987

*NETWORKRATES

Contents: The current rate schedule for the use of the Merit
               Computer Network.

Use: The rate schedule may be obtained via the $COPY command,
               e.g.,

                   $COPY *NETWORKRATES

302 *NETWORKRATES

January 1987

<div align="center">*OBJLIST</div>

Contents: The object-module listing program.

Purpose: To list the contents of an object module in a formatted
                 form for easy reading.

Use: This program is invoked by the $RUN command.

Program Key: *OBJLIST

Logical I/O Units Referenced:
                 SPRINT - output listing.
                 SERCOM - error comments to the user.
                 GUSER - input of file names if the PAR field is omitted.

Parameter: The PAR field of the $RUN command may contain the name of
                 a single file, plus applicable keywords, if no filenames
                 are given via the logical I/O unit GUSER.

Return Codes: Always zero.

Description: The output listing from this program gives the following
                 information:

                     (1) The name of the file being listed.
                     (2) The associated line number or sequential pointer
                          for each record in the module.
                     (3) The type of object module record indicated in
                          columns 2-4 of each record.
                     (4) The formatted list of each record.

                 If no PAR field is given on the $RUN command, the program
                 will prompt the user for a file name. The reply should
                 consist of a file name and any applicable forms of the
                 LIST keyword. These two elements should be separated by
                 one or more blanks. This keyword may be used to control
                 the selective listing of records from the file.   The
                 types of input records that may be specified by the LIST
                 keyword are: ESD, TXT, CSI, RLD, END, SYM, FTN, LDT,
                 REP, DEF, ENT, NCA, COM, MDL, LCS, LIB, RIP, and DIR.
                 These records are identified by characters 2 through 4 of
                 the record. All record types should be separated by
                 commas. To list all but certain types, the list of
                 loader records not desired should follow "LIST=ALL-".
                 For further details on loader input records, see the
                 section "The Dynamic Loader" in MTS Volume 5, System
                 Services.

                     The following examples illustrate the various forms of
                     the keyword:

                     LIST=ALL   List all records (the default).

                     LIST=TXT   List only TXT records.

                     LIST=ESD,END   List only ESD and END records.

                     LIST=ALL-TXT   List all records except TXT records.

                     LIST=ALL-TXT,CSI   List all records except TXT and CSI
                                          records.

                     Unless a PAR field is specified, an attention interrupt
                     will cancel the current module listing and the program
                     will prompt for another file name. If a PAR field is
                     given, the program will terminate at the point of the
                     interrupt.

                     In both batch and conversational mode, all input errors
                     generate an error message, but will not terminate the
                     program. The program will continue to prompt for file
                     names until an end-of-file is given.

                     The minimum and maximum output line lengths are 70 and
                     121 characters, respectively. A short format will auto-
                     matically be used for terminals with line lengths less
                     than 121 characters. For large files, the user should
                     copy the output to *PRINT* since the amount of output
                     will be great.

Examples: $RUN *OBJLIST PAR=OBJPROG

                     The above example will list the object modules in
                     the file OBJPROG on *SINK* (the default for SPRINT).

                 $RUN *OBJLIST SPRINT=*PRINT*
                 LIB LIST=ESD
                 MODULE LIST=ALL-TXT
                 $ENDFILE

                     In the above example, for the file LIB, only the ESD
                     records are listed; for the file MODULE, all records
                     except the TXT records are listed. All output is
                     written on *PRINT*.

January 1987

<u>*OBJSCAN</u>

Contents: The object-scanning program.

Purpose: To give an edited account of the contents of an object
                 file.

Use: This program is invoked by the $RUN command.

Program Key: *OBJSCAN

Logical I/O Units Referenced:
                 GUSER - input for file names if no PAR field is given.
                 SPRINT - output information from the program.
                 SERCOM - error comments to the user.

Parameter: The PAR field of the $RUN command may contain the name of
                 a single file plus applicable keywords if no file names
                 are to be read from GUSER.   The keywords must be
                 separated by at least one blank or comma.

Return Codes: Always zero.

Description: The object-scanning program reads an object file and
                 gives an edited account of the type of loader records in
                 the file.   If the file name is not specified in the PAR
                 field of the $RUN command, the program prompts for the
                 name of a file to be scanned. The reply, which must
                 consist of a file name and/or keywords separated from
                 each other by a comma and/or a blank, is read from GUSER.
                 Any file name which is identical to a keyword name (e.g.,
                 DATE) must be prefixed with the current MTS file-name
                 character, normally '#' (thus #DATE). If no file name is
                 specified, then specified keywords become   defaults.
                 Otherwise, the program will do the following while
                 scanning the specified file:

                     (1) Print out all entry point names in the ESD
                          records. Any blank entry point name will be
                          replaced by "(BLANK)" or "$PRIVATE", whichever is
                          applicable.
                     (2) Indicate the breaks between two kinds of records
                          (e.g., "TXT" or "RLD") by printing the beginning
                          and ending line numbers of a block of records
                          having the same characters in positions 2 through
                          4.   Example: "TXT"(3,45).   This is called the
                          LONG form.
                     (3) Indicate the beginning and ending line numbers of
                          object modules followed by a list of entry point
                          names.   Each module is assumed to have the first

record of type ESD, SYM, or FTN and the last
record of type END. All other records will be
ignored. This is the SHORT form usually useful
for object modules produced by the FORTRAN IV (H)
and the PL/I (F) compilers.
(4) Print out the date whenever an END record has the
date inserted in positions 33 through 72.

When all information has been given for the specified
file, the program prompts for another file name. The
program is terminated if the PAR field was specified or
when the user enters an end-of-file through GUSER.

Keywords:

FILE=FDname Specify the object file to be scanned.

MAXLEN=n   This controls the number of characters per
ORL=n   output line. The default is 121 or the
             maximum output device length, whichever is
             smaller.

MAXNAMES=n   Only at most "n" entry point names per
MAX=n   module will be printed. "n" must be in
             the range (0,256). The default is 256.

DATE   This prints the date information from the
D   END records.

NODATE   This suppresses the printing of the date
ND   information.

LONG   See the above description of the LONG form
L   in (2) above.

SHORT   See the above description of the SHORT
S   form in (3) above.

MTS   This returns control to MTS. A $RESTART
             command will restart the program.

The default keywords are: DATE, LONG, and MAXNAMES=256.
The values of all keywords remain in effect until
respecified by the user.

Examples: $RUN *OBJSCAN PAR=ALPHA

              In the above example, the program scans the file
              ALPHA and prints the information of the file con-
              tents on *SINK* (the default for SPRINT).

January 1987

*OBJUTIL

Contents: The MTS object-file editor.

Purpose: To edit files that contain programs in object-module form.

Use: The program is invoked by the $RUN command.

Program Key: *OBJUTIL

Logical I/O Units Referenced:
                SCARDS - either the input file containing the object
                         modules to be replaced or a sequence of
                         commands.
                SPRINT - printed output produced by the object-file
                         editor.
                SERCOM - diagnostic messages.
                GUSER - user responses in conversational mode.
                0   - default unit for the object file to be edited.

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command.   The parameters must be
                separated by commas or blanks. Parameters may be negated
                by "-", "~", "NO", or "N". The minimum acceptable
                abbreviation for each parameter is underlined.

                    BREAK=n

                        Each new module added to the edit file begins at
                        the next highest multiple of the line number "n".
                        Lines currently in the edit file are not changed
                        to reflect the new value. The default value is
                        1.000. The BREAK parameter has no effect for
                        sequential edit files.

                    COMSAVE

                        The COMSAVE parameter specifies that COM (com-
                        ment) records are to be saved during processing.
                        NOCOMSAVE specifies that COM records are not to
                        be saved. The default is NOCOMSAVE.

                    DIRECTORY

                        The DIRECTORY parameter specifies that the
                        object-file editor is to produce a two-record DIR
                        directory of the object modules in the edit file
                        for future use by the object-file editor. The
                        DIR record and its directory are written at line

numbers -1 and 0, respectively, in the edit file
when the object-file editor terminates proces-
sing.   This record will normally not be read by
the system loader since it is in the negative
line number range of the file. On subsequent
uses of the object-file editor, the program will
read this DIR record directory to build its
internal representation of the edit file instead
of reading the entire file as it must do if no
DIR record exists.   The presence of this DIR
record will increase the file space required for
the edit file but will greatly reduce the expense
of replacing object modules in an edit file
containing an object program consisting of a very
large number of modules. The default is NODIREC-
TORY, i.e., the object-file editor will not
produce a DIR record directory. However, if the
edit file contains a DIR record at lines -1 and
0, then DIRECTORY is assumed.   The DIRECTORY
parameter has no effect for sequential files.

DLR

If the DLR (delete library records) parameter is
specified, library (LIB and DIR) control records
will be deleted from the edit file.   NODLR
specifies that library records are not to be
deleted. The default is NODLR.

DMD

If the DMD (delete multiple definitions) parame-
ter is specified, object modules which are mul-
tiple definitions will be deleted from the edit
file. NODMD specifies that multiple definitions
are not to be deleted. The default is NODMD.

EMPTY

The EMPTY parameter specifies that the edit file
assigned to unit 0 is to be emptied before use.
NOEMPTY specifies that the edit file is not to be
emptied.   The default is NOEMPTY. EMPTY has no
effect in command mode.

FILL=xx

Two hexadecimal digits "xx" specify a character
to fill the gaps in the text of the control
sections. The default is FILL=81.

GAPSIZE=n

> "n" specifies the maximum object module text gap
> size to be filled during the object-file editor
> processing. The default is ORL/2 or 256, which-
> ever is smaller.

LIBRARY

> The LIBRARY parameter specifies that the edit
> file (if empty) is to be formatted as a loader
> library file.   In this case, the object-file
> editor will generate a DIR loader record at the
> beginning of the file and subsequently update the
> directory as object modules are added to the
> library. The default is NOLIBRARY, i.e., the
> file is not formatted as a loader library file.
> However, if the edit file is not empty, the
> object-file editor will examine the file and
> automatically determine whether or not it is a
> library file and act accordingly. Note that the
> LIBRARY parameter is distinct from the DIRECTORY
> parameter which requests the object-file editor
> to generate and maintain a special directory in
> the negative line-number range of the object file
> for its own use when editing.

MISCSAVE

> The MISCSAVE parameter specifies that miscel-
> laneous (ALI, RIP, DEF, OPT, NCA, and LCS) loader
> records are to be preserved during processing.
> NOMISCSAVE specifies that miscellaneous records
> are to be discarded. The default is MISCSAVE.

MSGSAVE

> The MSGSAVE parameter specifies that MSG (mes-
> sage) records are to be saved during processing.
> NOMSGSAVE specifies that MSG records are not
> saved. The default is MSGSAVE.

OPTIMIZE

> The OPTIMIZE parameter specifies that object
> modules are to be optimized, i.e., reformatted
> according to ORL, FILL, and GAPSIZE. NOOPTIMIZE
> specifies that input records of object modules
> are to be copied as is. The default is OPTIMIZE.

ORL=n

      "n" specifies the maximum output record length to
be used for output produced by the object-file
editor. The default is the maximum output record
length of the edit file (normally 32767).

QUIT

      The QUIT parameter specifies that if the object-
file editor encounters any errors in batch mode,
the user is signed off. NOQUIT specifies that
the batch user is not to be signed off if an
error occurs. The default is NOQUIT.

SLOTS=n

      "n" specifies the number of slots (one slot per
entry point) to be allocated for the DIR record
of a library file. "n" may be between 1 and
2730; the default is 128 for sequential files or
the minimum (the least possible number of slots)
for line files. The SLOTS parameter should be
specified only if the edit file is empty and is
to be formatted as a library file. Alternative-
ly, the LIBRARY parameter may be specified if the
empty edit file is to be a library file and, if a
sequential file, it requires no more than 128
slots.

SYMSAVE

      The SYMSAVE parameter specifies that SYM (symbol)
records are to be saved during processing. NO-
SYMSAVE specifies that SYM records are not saved.
The default is SYMSAVE.

TERSE/VERBOSE

      The TERSE/VERBOSE parameters control the amount
of information produced by the verification of
some commands. TERSE specifies that minimal
information is requested; VERBOSE specifies that
full information is desired. TERSE is an antonym
for VERBOSE. The default is the setting of the
MTS TERSE option. This parameter pair has no
effect if NOVERIFY is specified.

UPDATE/REPLACE

      The UPDATE/REPLACE parameters control whether the
file assigned to SCARDS is an update or replace-
ment file in no-command mode. UPDATE specifies

that object modules read from SCARDS not in the edit file will be included. REPLACE specifies that new modules will be excluded. The default is UPDATE.

VERIFY

The VERIFY parameter specifies that verification for each command is requested. NOVERIFY suppresses the verification. The default is VERIFY.

Return Codes: Always zero.

Description: The MTS object-file editor is used for the editing of object files. The object-file editor provides facilities to replace, add, delete, or correct an object module. Object modules that are added to the object file are automatically reformatted into maximum-sized records to reduce both the loading time and storage requirements. In addition, the object-file editor can generate and edit DIR-type loader library files.

Many of these same facilities are also provided by the MTS linkage editor described in this volume. However, since the linkage editor makes more general assumptions concerning the reformatting and optimization process, it generally is less efficient in providing these services. The object-file editor particularly is more efficient when replacing a single module in a file containing many modules.

Those users who want only to update their object-module file can simply issue the following command:

    $RUN *OBJUTIL SCARDS=inFDname 0=editFDname

In the above case, object modules are read from "inFD-name", converted to the optimized format, and written on "editFDname" replacing any previous definitions. The logical structure of the input modules is completely preserved with the following exceptions:

(1) The duplicate modules from "inFDname" are discarded.
(2) REP records of input modules are absorbed into the text of object modules.
(3) Nonabsolute DEF records are incorporated into the external symbol dictionary of the appropriate output modules.
(4) COM records are discarded.
(5) Records other than SYM, ESD, TXT, CSI, RLD, END, REP, and nonabsolute DEF, are placed in front of the output modules. An example is a NCA record.

A typical example is a FORTRAN program consisting of several subroutines whose object modules are in the file OBJ. While debugging the program, the user discovers an error in one of the subroutines. This error may be corrected in the source file by the MTS file editor. After the program is recompiled, the object-file editor may be used to replace the erroneous module in the object file. The last two stages would be as follows:

```
$RUN *FTN SCARDS=PROG(200,299) SPRINT=*PRINT*
$RUN *OBJUTIL SCARDS=-LOAD 0=OBJ
```

In this example, the source code for the subroutine is in the file PROG at the specified line range. The FORTRAN compiler places the new object module into the file -LOAD which is read by the object-file editor and replaces the old version of the subroutine in the file OBJ.

Those users who desire to use the more advanced features of the object-file editor must use the object-file editor command language which is described in MTS Volume 5, System Services. Commands are read from SCARDS and printed output is written on SPRINT. The typical $RUN command to use the object-file editor in command mode is

```
$RUN *OBJUTIL
```

Command input is terminated by an end-of-file or by the STOP command.

Examples: $RUN *OBJUTIL SCARDS=-LOAD 0=OBJPROG

The above example reads the object modules from the file -LOAD, converts them to an optimized format, and writes the result to the file OBJPROG replacing any previous definitions.

```
$RUN *OBJUTIL
EDIT OBJPROG
UPDATE FROM -LOAD
STOP
```

The above example, using the object-file editor command language, is equivalent to the first example.

January 1987

*OSMAC

Contents: The macro library for the IBM Operating System (OS).

Purpose: To enable OS users to assemble their programs in MTS.
                These programs must not be run under MTS.

Use: This file should be specified as a macro library when
                running the 360-assembler.

Program Key: *EXEC

Description: The references for the macro library are:

                (1) OS/360 System Supervisor Services and Macro In-
                     structions, form GC28-6646.
                (2) OS Data Management Macro Instructions, form
                     GC26-3794.

January 1987

<u>*OVERDRIVE</u>

Contents: The OVERDRIVE preprocessor.

Purpose: To allow the use of structured programming techniques in
                FORTRAN programs.

Use: The program is invoked by the $RUN command.

Program Key: *OVERDRIVE

Logical I/O Units Referenced:
                SCARDS - OVERDRIVE source.
                SPRINT - listing output.
                SERCOM - error messages (if in conversational mode).
                2   - target module output in standard FORTRAN.

Return Codes: 0 - Successful return.
                4 - Warning messages printed.
                8 - Errors detected.

Description: The OVERDRIVE preprocessor extensions to FORTRAN use
                structured programming concepts to allow the clearer
                expression of a program's flow of control. The target
                module generated by OVERDRIVE may be compiled either with
                the FORTRAN-G or FORTRAN-H compilers via the *FTN inter-
                face program.

                The complete description of the OVERDRIVE preprocessor is
                given in the section "OVERDRIVE" in MTS Volume 6, <u>FORTRAN
                in MTS</u>.

January 1987

*PAGEFONTCONVERT

Contents: The line-printer to Xerox 9700 page-printer reformatting
                program.

Purpose: To reformat output intended for a line printer to output
                appropriate to the Xerox 9700 page printer.

Use: The program is invoked by the $RUN command.

Program Key: *PAGEFONTCON

Logical I/O Units Referenced:
                SCARDS - input data to be reformatted.
                SPUNCH - reformatted output to be printed on the Xerox
                        9700 page printer.
                GUSER - prompting for program parameters.
                SERCOM - error messages.

Parameters: The four program parameters (described below) may be
                entered in the PAR field of the $RUN command. The
                parameter pairs must be enclosed in parentheses and
                separated by commas; omitted (defaulted) parameters must
                be indicated by a comma, e.g.,

                    $RUN *PAGEFONTCONVERT ... PAR=,(1,T),,(3,F)

Return Codes: Always zero.

Description: The *PAGEFONTCONVERT program reads output containing
                boldface and underscored text intended for a line printer
                and reformats it into output appropriate for the Xerox
                9700 page printer. The reformatted output takes advan-
                tage of the multiple-font capabilities of the page
                printer.

                With output intended for the line printer, a boldface
                character is printed by overprinting the character with
                itself, thus producing a darker copy of the character on
                the output. However, with the page printer, overprinting
                a character by itself will not produce a darker copy of
                that character. Instead, a different font must be used
                to achieve this effect.

                The *PAGEFONTCONVERT program divides the input text read
                from SCARDS into four categories:

                    (1) Normal characters (characters that are not in any
                         of the next three categories).
                    (2) Underlined characters (characters that are over-

> printed with the underscore character).
> (3)   Boldface characters (characters that are over-
>   printed with the same character).
> (4)   Underlined boldface characters (boldface charac-
>   ters that are also overprinted with the under-
>   score character).

Each of these categories can be assigned a font number
and an underline flag.   The font number specifies the
page-printer font index to be used for that category and
the underline flag (TRUE or FALSE) indicates whether text
in that category is to be underlined. Unless specified
in the PAR field of the $RUN command, the user will be
prompted for the font number and underline flag, i.e.,

> Normal characters   ?
> Underlined characters   ?
> Bold characters   ?
> Underlined bold characters ?

The response must be the font number followed by the
underline flag, e.g, 1,T or 2,T. Any font number may be
specified that is valid for the font package being used
for the page printer output.

The following defaults hold for each category:

> (1)   Normal = 1,F.   This specifies font 1 without
>   underlining.
> (2)   Underlined = 2,F. This specifies font 2 (italic
>   with most font packages) without underlining.
> (3)   Bold = 3,F. This specifies font 3 (bold with
>   most font packages) without underlining.
> (4)   Bold underlined = 3,T.   This specifies font 3
>   (bold with most font packages) with underlining.

Characters that are overprinted, but are not in cate-
gories 2, 3, or 4, are overprinted in the font specified
for category 1.

The *PAGEFONTCONVERT program processes the input data in
the following manner:

> (1)   The following *PAGEPR command is inserted at the
>   head of the output file:
>
> $**$ FONTINDEX=2
>
> (2)   The output data is shifted to the right by 1
>   column starting at column 2.   Column 1   is
>   retained as the logical carriage-control charac-
>   ter. The font index is inserted into column 2.

The output from *PAGEFONTCONVERT may be printed on the
Xerox 9700 page printer by using the *PAGEPR program.

Example: $RUN *PAGEFONTCONVERT SCARDS=TXT SPUNCH=-OUT
                EXECUTION BEGINS
                Enter one character (font number) and either "T" or
                "F" (whether underlining should occur), separated
                by a comma for each of the prompts following.
                Normal characters   ?
                Underlined characters   ?
                Bold characters   ?
                Underlined bold characters ? 4,T
                EXECUTION TERMINATED
                $RUN *PAGEPR SCARDS=-OUT PAR=PORTRAIT

                    In the above example (using the font package provid-
                    ed by PORTRAIT), characters in the normal category
                    are printed in font 1 without underlining (the
                    default),   underlined characters are printed in font
                    2 (italic) without underlining (the default),   bold
                    characters are printed in font 3 (bold) without
                    underlining (the default), and bold underlined char-
                    acters are printed in font 4 (script)   with
                    underlining.

                $RUN *PAGEFONTCONVERT SCARDS=TXT SPUNCH=-OUT PAR=,,,(4,T)
                $RUN *PAGEPR SCARDS=-OUT PAR=PORTRAIT

                    This example is the same as the first example,
                    except that the font number and underline flag are
                    specified in the PAR field of the $RUN command.

<u>*PAGEPR</u>

Contents: The Xerox 9700 page-printer output processing program.

Purpose: To print output on the Xerox 9700 page printer.

Use: The program is invoked by the $RUN command.

Program Key: *PAGEPR

Logical I/O Units Referenced:
                SCARDS - text to be printed (each line is assumed to
                         contain a logical carriage-control character in
                         column 1).   Control lines beginning with $9700
                         specify new options rather than text to be
                         printed. These new options (with one exception)
                         will take effect the next time a new image is
                         started. See below for the list of options.

Parameters: The PAR field should contain the options that are to be
                processed before lines are read from SCARDS. These
                options will take effect on the first page of the output.
                See below for the list of options.

Options: Options are specified either in the PAR field of the $RUN
                command or by giving them in the SCARDS input, on lines
                beginning with $9700.   In both cases, if more than one
                option is given, the options should be separated by
                commas or blanks.   Since each $9700 line is processed
                independently, the ordering of some options may be
                important. FORMAT (or equivalent) must appear before the
                FONTLIST option since it resets the font list. PAGEORI-
                GIN must appear after the FORMAT and FONTLIST options
                since it depends on the spacing of the first font. The
                options are described in Reference R1038, "Using the
                Xerox 9700 Page Printer."

Description: The *PAGEPR program is the older method of printing
                output on the Xerox 9700 page printer. Before the
                installation of the Resource Manager, this program was
                required for all printing that used fonts other than the
                default landscape format.

                Two versions of this program are still available. The
                first version simply passes the print file onto the
                Resource Manager for processing without intervention.
                This version is provided in order to maintain compatibil-
                ity with MTS command macro files and $SOURCE files that
                include calls to *PAGEPR. This version is run by issuing
                the command

| $RUN *PAGEPR INPUT=file PAR=options
|
| "file" is the name of the file containing the text you
| want printed. The PAR field should contain the various
| printing options that might apply to your file.
|
| The second version actually processes the print file in
| the same manner as the previous version of *PAGEPR under
| HASP. This version is available in order to provide some
| of the more esoteric options that are not supported by
| the Resource Manager, such as the third and fourth
| parameters of the PAGEORIGIN $9700 command and the full
| implementation of the LPP command. This version is run
| by issuing the command
|
| $RUN *PAGEPR INPUT=file PAR=!
|
| or
|
| $RUN *PAGEPR INPUT=file PAR=!options
|
| Note that in the PAR field, any options must follow the
| "!" without intervening blanks (e.g., PAR=!PORTRAIT).

January 1987

<u>*PASCALJB</u>

Contents: A Pascal Compiler from Plug Compatible Software, Inc.

Purpose: To compile Pascal/JB source programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:
               SCARDS - Pascal source-program input.
               SPRINT - listing output including compiler diagnostics.
               SPUNCH - object deck output.
               SERCOM - compiler diagnostics.
               0   - object library to add (or replace) module to.
               1 to 8 - include library

Return Codes: 0 - Successful return.
               4 - Compilation warnings.
               8 - Compilation errors.

Description: *PASCALJB is a Pascal compiler that supports superset of
               the Pascal programming language developed by Nicholas
               Wirth in the late 60s. This compiler conforms to the
               language accepted by IBM's Pascal/VS compiler; in addi-
               tion, it supports other features. Source programs writ-
               ten in Pascal/VS can be compiled and run under PASCAL/JB.

               The complete description of Pascal/JB is given in MTS
               Volume 20, <u>Pascal in MTS</u>. A brief summary of the
               compiler options is given below.

Compile-Time Options

Option	Default
CHECK / NOCHECK	CHECK
CLG / DECK	See Vol. 20
DEBUG / NODEBUG	DEBUG
DISPOSECHECK / NODISPOSECHECK	DISPOSECHECK
INITCHECK / NOINITCHECK	INITCHECK
ARRAYINIT / NOARRAYINIT	ARRAYINIT
RECORDINIT / NORECORDINIT	RECORDINIT
LANGLVL={STANDARD|EXTENDED|PVS}	LANGLVL=EXTENDED
LIBRARY=FDname	None
LINECOUNT=n	LINECOUNT=60
MARGINS=(m,n)	MARGINS=(1,255)
OPTIMIZE / NOOPTIMIZE	NOOPTIMIZE
PAGEWIDTH=n	PAGEWIDTH=132
SHORTRECORDS / NOSHORTRECORDS	SHORTRECORDS
SIZE=n	SIZE=4P
SOURCE / NOSOURCE	See Vol. 20
STATS	STATS
WARNING / NOWARNING	WARNING
XREF / NOXREF	NOXREF

Execution-Time Options:

DEBUG	Disabled
STACK=n	See Vol. 20
HEAP=n	HEAP=1P

January 1987

<div align="center">*PASCALTIDY</div>

Contents: The Pascal source program formatter.

Use: To reformat a Pascal source program for better
                readability.

Program Key: *PASCALTIDY

Logical I/O Units Referenced:
                SCARDS - Pascal source to be reformatted.
                SPUNCH - Reformatted source program.

Return Codes: 0 - Successful return.
                8 - Formatting unsuccessful.

Description: Pascaltidy is a Pascal program formatter that creates a
                copy of the source program, adding spacing and indenta-
                tion in appropriate ways as to make the program source
                file easier to read. Such "tidying" aids in debugging
                programs that compile correctly, but need to be proofread
                closely to detect logical errors.   It also helps make
                later alterations and additions to a program easier to
                perform.

                The Pascaltidy program is completely described in MTS
                Volume 20, Pascal in MTS.

January 1987

*PASCALVS

Contents: The IBM Pascal/VS Compiler.

Purpose: To compile Pascal/VS source programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *PASCALVS

Logical I/O Units Referenced:
                SCARDS - Pascal source-program input.
                SPRINT - listing output including compiler diagnostics.
                SPUNCH - object deck output.
                SERCOM - compiler diagnostics.
                0   - object library to add (or replace) module to.
                1 to 8 - include libraries

Return Codes: 0 - Successful return.
                8 - Error in compilation phase.
                12 - Error in translation/optimization phase.
                16 - Error in translation/optimization or run-time
                     environment.

Description: *PASCALVS contains a version of the IBM Pascal/VS compil-
                er that has been modified for use in MTS.

                For additional information, see the IBM publication,
                Pascal/VS Language Reference Manual, form SH20-6168, and
                MTS Volume 20, Pascal in MTS. A brief summary of the
                compiler options is given below.

Compile-Time Options

        Option   Default

    CHECK / NOCHECK   CHECK
    DEBUG / NODEBUG   DEBUG
    GOSTMT / NOGOSTMT   GOSTMT

    LANGLVL={STANDARD|EXTENDED}   LANGLVL=EXTENDED
    LINECOUNT=n   LINECOUNT=60
    LIST / NOLIST   NOLIST

    MARGINS=(m,n)   MARGINS=(1,100)
    OPTIMIZE / NOOPTIMIZE   NOOPTIMIZE
    PAGEWIDTH=n   PAGEWIDTH=132

    PXREF / NOPXREF   NOPXREF
    SEQ=(m,n) / NOSEQ   NOSEQ
    SOURCE / NOSOURCE   See Vol. 20

    WARNING / NOWARNING   WARNING
    XREF / NOXREF   NOXREF

Execution-Time Options

    COUNT   Disabled
    DEBUG   Disabled
    ERRCOUNT=n   ERRCOUNT=1

    NOCHECK   Disabled
    NOSPIE   Disabled
    STACK={nK|nP}   STACK=12K

    HEAP={nK|nP}   See Vol. 20
    SETMEM   Disabled

January 1987

<div align="center">

*PDSTOMNET, *MNETTOPDS

</div>

Contents: The Plot Description System (PDS) to Merit Computer
                Network plot-file conversion programs.

Use: The programs are invoked by the $RUN command.

Program Key: *PDSTOMNET
                *MNETTOPDS

Logical I/O Units Referenced:
                GUSER - input plot-file names.
                SERCOM - prompting messages.
                SPUNCH - output plot files produced.

Parameters: The name of a single plot file to be converted may be
                specified in the PAR field of the $RUN command.

Return Codes: Always zero.

Description: With the *PDSTOMNET program, if the user has not speci-
                fied a plot-file name in the PAR field of the $RUN
                command, the program prompts the user with

                    ENTER PLOT FILE NAME

                After the file name has been entered, the plots contained
                in the file are converted to the Merit Computer Network
                standard plot-file format (5(F8.3,F7.3,I1)). If the PAR
                field was not specified, the user is again prompted for a
                plot-file name. An end-of-file terminates execution of
                the program.

                For the *MNETTOPDS program, if the user has not specified
                a plot-file name in the PAR field of the $RUN command,
                the program prompts the user with

                    ENTER MERIT NETWORK PLOT FILE NAME:

                After the file name has been entered, the plots contained
                in the file are converted to the Plot Description System
                format. If the PAR field was not specified, the user is
                again prompted for a plot-file name. An end-of-file
                terminates the program.

                For additional information, contact the Merit Computer
                Network Office.

Examples: $RUN *PDSTOMNET SPUNCH=MNPLOT PAR=PDSPLOT

                   In the above example, the plot or plots in the file
                   PDSPLOT are converted to the Merit Computer Network
                   plot-file format and written to the file MNPLOT.

          $RUN *MNETTOPDS SPUNCH=PDSFILE PAR=MNFILE

                   In the above example, the plot or plots in the file
                   MNFILE are converted to the PDS plot-file format and
                   written to the file PDSFILE.

January 1987

*PEXIT

Contents: An Assembler-H listing postprocessor.

Purpose: To make cosmetic alterations to the source program
               listing generated by *ASMH. This will provide a more
               readable listing, especially if the structured program-
               ming macros are used.

Use: The subroutine is invoked by the H-level Assembler by
               specifying the PEXIT parameter, e.g.,

                   $RUN *ASMH [I/O units] PAR=PEXIT=*PEXIT

Program Key: *PEXIT

Description: The complete description of this subroutine is given in
               MTS Volume 14, 360/370 Assemblers in MTS.

January 1987

*PFORT

Contents: The PFORT Verifier.

Purpose: To check FORTRAN programs for adherence to PFORT, a
                 portable subset of the 1966 version of American National
                 Standard FORTRAN.

Use: The program is invoked by the $RUN command.

Program Key: *PFORT

Logical I/O Units Referenced:
                 SCARDS - Input to the Verifier. This should take the
                          form of one or more FORTRAN source program units
                          to be verified. If global analysis is request-
                          ed, the sequence of source program units should
                          comprise a single executable FORTRAN program.
                          An input line containing a period in column 1
                          will cause the Verifier to stop processing
                          input. However, the preferred method of speci-
                          fying the end of the input is to use one of the
                          usual end-of-file mechanisms supported by MTS.

                 SPRINT - Output from the Verifier. This may be program
                          listings, messages indicating PFORT violations,
                          symbols tables, cross references, or global
                          analysis.

Return Codes: Always zero.

Parameters: The following parameters or their negations may be
                 specified in the PAR field of the $RUN command. Parame-
                 ters must be separated by blanks or commas. The minimum
                 acceptable abbreviation for each parameter is underlined.
                 The parameters are treated from left to right, so in case
                 of conflicting parameters, the rightmost one will be put
                 into effect. Any parameter may be negated by prefixing
                 it with NO, N, -, or ~. All of the parameters below are
                 active by default.

                 LIST   Print a source listing for each program unit.

                 SYMBOLS   Print a symbol table for each program unit.

                 XREF   Print cross references with each symbol table.
                           This option will be honored only if SYMBOLS is
                           active.

GLOBAL   Perform and print a global analysis of the
          input source program. The Verifier itself may
          deactivate this option if it encounters serious
          errors during the processing of the input.
          Once deactivated, whether by the user or by the
          Verifier, the GLOBAL option cannot be
          reactivated.

Parameters may also be specified by including a source
line with a C in column 1 and an asterisk in column 2,
followed by parameters anywhere in columns 3 to 72. This
permits the user to turn the output options on or off
locally.

Example: C*   LIST
               SUBROUTINE A
                ...
               END
          C*   NOLIST
               SUBROUTINE B
                ...
               END
          C*   LIST
               SUBROUTINE C
                ...

In this example, selected program units are LISTed. Such
local parameters embedded in the source may be safely
passed on to any of the FORTRAN compilers, which will
treat them as ordinary FORTRAN comments.

Description: The PFORT Verifier produces a statement by statement
              listing of each program unit, followed by a symbol table
              which lists attributes and cross references for all the
              symbols in that program unit.

              If a violation of the PFORT standard is discovered, an
              error message, preceded by three asterisks, is printed
              directly below the statement at which the error occurred.
              If the NOLIST option is in effect, the error message is
              accompanied by the program unit name and internal state-
              ment number of the statement in error.

              The global structure of the program is presented in an
              alphabetized list of all its program units, showing for
              each program unit its arguments, its common blocks, the
              program units it calls, and the program units which call
              it. A table of global common block definitions is
              produced.

              For the complete description of the PFORT Verifier, see
              Computing Center Memo 406, "*PFORT."

January 1987

<u>*PLC</u>

Contents: The Cornell compiler for PL/I.

Purpose: To compile and execute PL/I programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *PLC

Alt. Name: *PL/C

Logical I/O Units Referenced:
                 SCARDS - The source program to be compiled and its
                          standard input unit.
                 SPRINT - The compiler listings and diagnostics and the
                          source program's standard output unit.

                 All logical I/O units can be referred to in the source
                 program, either by using the logical I/O unit as a file
                 name (e.g., SCARDS) if it is alphabetic or by using a
                 TITLE option (e.g., TITLE('SCARDS') or TITLE('0')) on an
                 OPEN statement.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command.   The parameters must be
                 separated by a comma. Abbreviations, where permissible,
                 are underlined.

                 <u>DEFAULTS</u>=list
                          Define the default compiler options which will
                          be set at the beginning of each PL/C program to
                          be compiled.   Any of the compilation options
                          described in the section "PL/C" in MTS Volume
                          7, <u>PL/I in MTS</u>, may appear on the right-hand
                          side of this keyword. If more than one option
                          is to be specified, they must be enclosed
                          within parentheses, and separated with commas.
                          A listing of the default parameters will appear
                          at the beginning of the compilation if the
                          OPTLIST compiler option is in effect.

                 HARDSTOP Should not be used except for those persons
                           maintaining the PL/C compiler.

                 <u>JCL</u>   Permit JCL "//..." cards to be recognized by
                           PL/C. The default case is not to treat them in
                           any special manner.

                    STAT   Print compilation and execution statistics on
                    NOSTAT   SERCOM if PL/C is being run in conversational
                               mode and compiler output is being diverted to a
                               file/device other than the terminal. STAT is
                               the default option.

                    SIZE=n   Define the amount of virtual memory to be
                               acquired by *PLC for the PL/C program code and
                               dynamic storage allocation during execution.
                               The value given specifies the region size in K
                               (multiples of 1024) bytes, 20≤value≤1024.   The
                               default is SIZE=50.

Return Codes: Always zero.

Description: PL/C is a compiler designed for instruction in the PL/I
               language. It provides high-speed compilation, reasonably
               efficient execution, excellent diagnostic assistance, and
               upward compatibility with the IBM PL/I-F compiler.

               Due to its design goals, the full PL/I language is not
               supported, but the PL/C subset is large. For further
               information about PL/C and reference material concerning
               the MTS implementation of PL/C, see the section "PL/C" in
               MTS Volume 7, PL/I in MTS.

January 1987

<u>*PLOTSEE</u>

Contents: The plot-previewing program.

Purpose: To allow users with Plot Description System (PDS) plot
               files generated via *PLOTSYS to preview their plots on
               any graphic device supported by the Integrated Graphics
               (IG) system.

Use: The program is invoked by the $RUN command.

Program Key: *PLOTSEE

Logical I/O Units Referenced:
               SCARDS - user input.
               SPRINT - program nonplot outut.
               0   - input plot file.
               9   - plot output for CalComp plotter.

Return Codes: 0 - Successful return.
               8 - Unit 0 unassigned.
               12 - Invalid plotfile.
               16 - Internal error in program.

Description: The *PLOTSEE program is a graphics program that displays
               plots from a plot file on a graphics terminal, providing
               a simple and fast way to check plot files for accuracy
               before submitting them to the plot queue. A plot file is
               used on MTS to hold output for a picture that is to be
               plotted on the CalComp plotters. The *IG, *TELLAGRAF,
               and *PLOTSYS graphics packages can all produce plot
               files.

               A plot file contains encoded instructions for drawing one
               or more plots on the CalComp plotter. These encoded
               instructions need to be stored so a program can be run
               which will interpret them and have the CalComp plotter
               draw the picture or pictures represented in the plot
               file. The program that submits plot files to be plotted
               is *CCQUEUE, which is described in this volume.

               *PLOTSEE can be used to serve a number of functions, from
               previewing a plot before spending the time and money to
               get the actual plot, to manipulating several plots at one
               time to form a composite plot. *PLOTSEE's enlargement
               facility allows you to see a detailed view of a section
               of a plot. *PLOTSEE also allows you to move, scale,
               rotate and combine plots into a single picture, to which
               lines, text and other features may be added. Any plot,
               enlargement, or any picture which is a combination of

plots and enlargements that *PLOTSEE is able to produce can be put into a new plot file for later display or plotting. If a remote plotter is attached to a terminal, *PLOTSEE can be told to draw the picture on either the terminal screen or the remote plotter.

The complete description of the *PLOTSEE program is given in Computing Center Memo 456, "*PLOTSEE."

Commands: A summary of the *PLOTSEE command language is given below. Commands may be given either interactively or in the PAR field of the $RUN command.

     ACTIVATE number   Sets the current picture.

     AUXPLOT plotter   Plots the screen image on a remote plotter.

     BLOWUP   Enlarges a section of the current plot.

     BOX   Draws a box around the current picture.

     COLOR color   Sets the color of the current picture.

     CONTINUE   Continues with the next frame.

     DEFAULT string   Changes default command sequence.

     DELETE   Deletes the current picture from the screen.

     DESTROY   Destroys unnamed pictures on a DISPLAY.

     DISPLAY picture   Displays only the indicated picture.

     DOWN amount   See the DY command.

     DRAW   Updates the screen.

     DUMP   Prints information about the data structure.

     DX amount   Displaces the current picture horizontally.

     DY amount   Displaces the current picture vertically.

EDIT plotfile	Changes the plot input file to plotfile.
ENGLISH	Specifies that units should be inches.
EXPLAIN command	Explains the use of a command.
FONT fontname	Sets the font to be used by the TEXT command.
FTNCMD command	Issues a command to the FORTRAN I/O monitor.
GET name	Makes a picture the current picture.
GLOBAL	Makes the screen image the current picture.
HELP	Enter the help facility.
IGCTRL switch	Turns KEEP switch on and off.
INVISIBLE	Makes the current picture invisible.
KEEP	Prevents DELETE from destroying pictures.
LEFT amount	See the DX command.
LINE	Draws a line on the screen.
LIST	Prints a list of the pictures currently defined.
MESSAGES	Prints information about the current picture.
METRIC	Makes all units metric.
MOVE	Moves the current picture to a new position.
MTS	Returns to MTS command mode.
NAME name	Assigns a name to the current picture.
NEXT	Displays the next picture.
NOPROMPT	Turns off prompting.

          PEN number   Sets the line type or color.

          PICK   Makes a displayed picture current.

          PICTURE   Sets the current picture.

          PLOT   Puts the screen image on a plotfile.

          POLYGON sides   Draws a polygon of "sides" sides.

          PPNUMBER   Prints the current picture number.

          PREFIX string   Specifies a command string to be
                                executed when a CONTINUE command is
                                executed.

          PRINT string   Prints "string" on SPRINT.

          PROMPT   Enables prompting.

          REDRAW   Redraws the screen.

          RIGHT amount   See the DX command.

          ROTATE degrees   Rotates the current picture.

          SCALE factor   Scales the current picture.

          SEGMENT   Picks part of a picture as the cur-
                                rent picture.

          SELECT   Allows previewing plots in any order.

          STOP   Terminates the run.

          UNDO   Undoes all the transformations on the
                                current picture.

          UP amount   See the DY command.

          VIEWPORT picture   Puts the indicated picture in a
                                viewport.

          VISIBLE   Makes the current picture visible.

          WINDOW   Displays a window of the current
                                picture.

          $mts-command   Executes an MTS command.

*PLOTSTACK

Contents: The program to stack plots vertically.

Purpose: To combine several plots by stacking them along the
                y-axis of the paper in order to save plotting paper.

Use: This program is invoked by the $RUN command.

Program Key: *PLOTSTACK

Logical I/O Units Referenced:
                SCARDS - input plot file of plots to be stacked.
                SPUNCH - output plot file of stacked plots ready for
                         *CCQUEUE processing.
                SERCOM - error messages.

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command:

                YMARGIN=n

                    The YMARGIN parameter specifies the amount of space
                    (in inches) to separate adjacently stacked plots.
                    The default is 2 inches.

                MAXPLOTS=n

                    The MAXPLOTS parameter specifies the maximum number
                    of plots that may be placed in a single stack. The
                    default maximum is 100 plots. Since the paper is
                    only 33 inches wide, the actual number of plots that
                    can be stacked is normally 2 or 3.

                SCALE=n

                    The SCALE parameter specifies a scale factor to be
                    applied to all plots before stacking is performed.

Return Codes: 0 - Successful return.
                8 - Fatal error (message printed on SERCOM).

Description: *PLOTSTACK reads in input plot file and generates stacks
                of plots in the output plot file. The user must then run
                *CCQUEUE to plot the output plot file.

                *PLOTSTACK stacks plots in the order in which they appear
                in the input plot file. The program places successive
                plots one above the other in the y-direction until it
                encounters a plot which

      (1) would make the stack higher than paper width
           allows,
      (2) would cause the MAXPLOTS value to be exceeded,
      (3) uses pens that may not be used in combination
           with pens used by the previous plots in the
           stack, or
      (4) requires a different scale factor (this generally
           would occur only if PLTSIZ was called for plots
           generated by the Plot Description System).

A new stack is started when any of the four above
conditions occurs.

Examples: $RUN *PLOTSTACK SCARDS=P1+P2 SPUNCH=STACK PAR=SCALE=.5

In the above example, the plots in the files P1 and
P2 are stacked into the file STACK. The plots are
scaled down to one half of their original size with
the default 2-inch spacing between plots.

| $RUN *PLOTSTACK SCARDS=IN SPUNCH=OUT
| PAR=YMARGIN=1,MAXPLOTS=3

In the above example, the plots in the file IN are
stacked into the file OUT leaving a 1-inch space
between plots and stacking at most 3 plots per
stack.

January 1987

<u>*PLOTSYS</u>

Contents: The University of Michigan Plot Description System sub-
               routine library.

Purpose: To enable users to generate file descriptions of pictures
               for input to graphical devices such as the Computing
               Center's digital plotter.

Use: *PLOTSYS should be concatenated to the files containing
               the user's object modules on the $RUN command, e.g.,

                   $RUN object+*PLOTSYS

Program Key: *EXEC

Logical I/O Units Referenced:
               9   - The MTS file or magnetic tape onto which plot
                         descriptions are to be written. This unit may
                         be changed via the PDS subroutines PLDNO or
                         PFLNAM.

Description: *PLOTSYS is a collection of FORTRAN-callable subroutines
               for drawing graphs and other types of pictures. These
               subroutines produce a device-independent plot description
               of the plot as output. To actually obtain the plot, this
               description must be placed into a queue for processing by
               the system postprocessor. See the description of *CC-
               QUEUE in this volume for instructions on queuing the plot
               description.

               The plot description may be written to a file or magnetic
               tape.   In the case of a tape, the tape may be labeled
               and/or blocked and positioned to any file.

               See MTS Volume 11, <u>Plot Description System</u>, for a
               complete description of the subroutines in *PLOTSYS. See
               MTS Volume 17, <u>Integrated Graphics System</u>, for descrip-
               tions of other graphics subroutines.

| ***PLUS, *PLUS11**

Contents: The PLUS programming language compiler.

Use: The compiler is invoked by the $RUN command.

| Program Keys: *PLUS and *PLUS11

Logical I/O Units Referenced:
                SCARDS - PLUS source program.
                SPRINT - program listing produced.
                SPUNCH - object module produced.
                SERCOM - error messages produced.
                0   - source program library.

Return Codes: 0 - Successful return.
                4 - Warnings issued.
                8 - Errors detected.

Description: Alan Ballard and Paul Whaley of the University of British
                Columbia have developed a language called PLUS, which is
                similar to Pascal but designed to be used for system
                implementation.

                PLUS is based to a large extent on the Sue system
                language, which was developed at the University of
                Toronto, circa 1971, for the specific purpose of imple-
                menting an operating system for the IBM 360 computers.
                The system was never completed; however compilers for
                both the 360 and PDP-11 were produced. The language was
                used with considerable success at Toronto (most noticeab-
                ly for the SP/k PL/I subsets project), and by a few
                students for thesis projects at UBC. The Sue language
                was derived (particularly in its data structure facili-
                ties) from Pascal.

                PLUS is superficially quite different from Sue or Pascal;
                however the underlying language semantics are really very
                similar.   Users familiar with the programming language C
                will also recognize much of its structure and semantics
                in PLUS.

                The current documentation for PLUS assumes some knowledge
                of PASCAL or similar higher level languages and is not
                easy reading for a novice. The manual (approximately 125
                pages long) is in the file PLUS:MANUAL, which can be
                copied directly to *PRINT*.   Several changes to the
                language and compiler are not documented in this manual.
                The file PLUS:NEWS describes these changes. PLUS:NEWS is
                required reading before actually using the language,

because there are some changes to the way programs
interface to the $RUN command.

At this time, there is no consulting available to help
users with problems with either the language, the compil-
er, or programs written in PLUS. The PLUS compiler,
however, is very reliable and is used heavily by the
Computing Center Staff. If you want to try writing
programs in PLUS, you should also print a copy of
PLUS:LIBRARY.DOC, which describes the source and object
libraries used by PLUS programs. This document is about
120 pages long. Also note that PLUS is reliably avail-
able only on MTS systems at this time.

A PLUS conference using *FORUM now exists. To become a
member

        $RUN *FORUM PAR=PLUS

or issue the FORUM command:

        JOIN PLUS

from within *FORUM.

| The version of PLUS in the file *PLUS produces code for
| programs run under MTS. The version of PLUS in the file
| *PLUS11 produces code for programs run on the PDP-11
| computer.

January 1987

<div align="center">

*PL1F

</div>

Contents: The MTS version of the IBM OS/360 PL/I(F) compiler.

Purpose: To compile PL/I source programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *PL1F

Alt. Name: *PL1

Logical I/O Units Referenced:
                 SCARDS - PL/I source program input (maximum input length
                      $\leq$ 100).
                 SPRINT - listing output, including diagnostics.
                 SPUNCH - macro deck and object deck output.
                 SERCOM - diagnostics (optional).
                 0   - object module output.

Return Codes: 0 - Successful return.
               4 - Warning messages printed.
               8 - Errors detected.
               12 - Severe errors detected.
               16 - Unrecoverable errors detected.

Description: The MTS PL/I compiler is derived from the IBM OS/360
               F-level PL/I compiler (version 5.5) with modifications
               for MTS features.

               The file *PL1FLIB contains the   subroutine   library
               required for running PL/I(F) programs. This file should
               be concatenated on the $RUN command to the file contain-
               ing the compiled PL/I program, e.g.,

                    $RUN object+*PL1FLIB

               For further information on compiling and running PL/I
               programs, see MTS Volume 7, PL/I in MTS. A brief summary
               of the compiler options is given below.   Abbreviations
               are indicated by underlining.

Option	Default
<u>A</u>TR / NOATR or NA	See Vol. 7
<u>C</u>HAR60 or C60 / CHAR48 or C48	CHAR60
<u>C</u>OMP / NOCOMP or NC	COMP
<u>D</u>ECK / NODECK or ND	DECK
DIAG / NODIAG	See Vol. 7
<u>EB</u>CDIC / <u>TT</u>Y / <u>B</u>CD	EBCDIC
EXTDIC or ED / NOEXTDIC or NED	NOEXTDIC
<u>E</u>XTREF / NOEXTREF or NE	NOEXTREF
FLAGW or FW / FLAGE or FE / FLAGS or FS	FLAGW
<u>F</u>REE / SORMGIN=(m,n[,c]) or SM=(m,n[,c])	SM=(1,72)
LINECNT=n or LC=n	LINECNT=60
<u>L</u>IST / NOLIST or NL	NOLIST
LOAD or LD / NOLOAD or NLD	NOLOAD
MACDCK or MD / NOMACDCK or NMD	NOMACDCK
<u>M</u>ACRO / NOMACRO or NM	NOMACRO
MTS / OS	MTS
NEST or NT / NONEST or NNT	NEST
NUM / NONUM	NUM
OPLIST or OL / NOOPLIST or NOL	See Vol. 7
<u>OP</u>T=n	OPT=1
SIZE={2K|nP}	SIZE=4P
<u>S</u>OURCE / NOSOURCE or NS	See Vol. 7
SOURCE2 or S2 / NOSOURCE2 or NS2	See Vol. 7
<u>S</u>TMT / NOSTMT or NST	STMT
SYNCHKT or SKT / SYNCHKS or SKS / SYNCHKE or SKE	SYNCHKS
<u>X</u>REF / NOXREF or NX	See Vol. 7

January 1987

<div align="center">*PL1OPT</div>

Contents: The MTS PL/I Optimizing Compiler.

Purpose: To compile PL/I source programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *PL1OPT

Logical I/O Units Referenced:
                 SCARDS - PL/I source-program input (maximum input length
                          ≤ 100).
                 SPRINT - listing output.
                 SPUNCH - macro-deck and object-deck output.
                 SERCOM - output for the TERMINAL option.
                 0   - object-module output.
                 1-19   - macro libraries for %INCLUDE statements.

Return Codes: 0 - Successful return.
                 4 - Warning messages printed.
                 8 - Errors detected.
                 12 - Severe errors detected.
                 16 - Unrecoverable errors detected.

Description: The MTS PL/I Optimizing compiler is derived from the IBM
                 OS PL/I Optimizing Compiler (release 3.1) with modifica-
                 tions for MTS.

                 The file *PL1OPTLIB contains the subroutine library
                 required for programs compiled by the PL/I Optimizing
                 Compiler. This file should be concatenated on the $RUN
                 command to the file containing the compiled PL/I program,
                 e.g.,

                     $RUN object+*PL1OPTLIB

                 Note: Programs compiled by the PL/I(F) and the PL/I
                 Optimizing compilers cannot be mixed.

                 The complete description of the PL/I Optimizing compiler
                 may be found in the next edition of MTS Volume 7, PL/I in
                 MTS.   A brief summary of the compiler options is given
                 below. Abbreviations are indicated by underlining.

Option	Default
AGGREGATE / NOAGGREGATE or NAG	NOAGGREGATE
ATTRIBUTES[({FULL|SHORT})] / NOATTRIBUTES or NA	See Vol. 7
CHARSET([{48|60}] [{EBCDIC|BCD}]) or	
CS([{48|60}] [{EB|B}])	CS(60 EB)
COMPILE / NOCOMPILE[({W|E|S})] or NC[({W|E|S})]	NOCOMPILE(S)
COUNT or CT / NOCOUNT or NCT	NOCOUNT
DECK / NODECK or ND	DECK
DUMP / NODUMP or NDU	NODUMP
ESD / NOESD	NOESD
FLAG[({I|W|E|S})]	See Vol. 7
FLOW[(n,m)] / NOFLOW	NOFLOW
GONUMBER or GN / NOGONUMBER or NGN	NOGONUMBER
GOSTMT or GS / NOGOSTMT or NGS	GOSTMT
IMPRECISE / NOIMPRECISE or NIMP	NOIMPRECISE
INCLUDE / NOINCLUDE or NINC	NOINCLUDE
INSOURCE or IS / NOINSOURCE or NIS	See Vol. 7
INTERRUPT / NOINTERRUPT or NINT	NOINTERRUPT
LINECOUNT(n) or LC(n)	LINECOUNT(60)
LIST[(m[,n])] / NOLIST	NOLIST
LMESSAGE or LMSG / SMESSAGE or SMSG	See Vol. 7
MACRO / NOMACRO or NM	NOMACRO
MAP / NOMAP	NOMAP
MARGINI('c') or MI('c') / NOMARGINI or NMI	MARGINI('|')
NEST / NONEST	NEST
NUMBER / NONUMBER or NNUM	NONUMBER
OBJECT / NOOBJECT or NOBJ	NOOBJECT
OFFSET / NOOFFSET or NOF	NOOFFSET
OPTIMIZE({TIME|0|2}) / NOOPTIMIZE or NOPT	NOOPTIMIZE
OPTIONS / NOOPTIONS or NOP	See Vol. 7
SEQUENCE(m,n) / NOSEQUENCE or NSEQ	NOSEQUENCE
SIZE({[-]nnnnn[{K|P}]|MAX})	SIZE(50P)
SOURCE / NOSOURCE or NS	See Vol. 7
STMT / NOSTMT	STMT
STORAGE or STG / NOSTORAGE or NSTG	NOSTORAGE
SYNTAX / NOSYNTAX[({W|E|S})]} or NSYN[({W|E|S})]	NOSYNTAX(S)
TERMINAL[(optlist)] / NOTERMINAL or NTERM	See Vol. 7
XREF[({FULL|SHORT})] / NOXREF or NX	See Vol. 7

January 1987

*PL1SCAN

Contents: The IBM PL/I source-program scanner with a modified
                 interface for operation under MTS.

Purpose: To provide a rough prescan (or syntax check) of programs
                 written in PL/I.

Use: The program is invoked by the $RUN command.

Program Key: *PL1SCAN

Alt. Name: *PL/1SCAN

Logical I/O Units Referenced:
                 SCARDS - source program to be scanned.
                 SERCOM - error messages from the scanner.
                 SPRINT - listing of the input source program.
                 SPUNCH - file to which the source is written.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command or in the %PROCESS batch card.
                 The parameters must be separated by a comma or by one or
                 more blanks.

                 SOURCE   prints the source on SPRINT (this defaults
                 NOSOURCE   to SOURCE in batch only).

                 CHAR48   specifies the 48-character set.
                 CHAR60   specifies the 60-character set (the
                                 default).

                 SORMGIN=(m,n)   specifies the left-hand (m) and right-hand
                                 (n) source margins. These default to 1
                                 and 72, respectively. The right-hand mar-
                                 gin must be equal to or less than 100.

                 The parameters SOURCE, NOSOURCE, CHAR48, CHAR60, and
                 SORMGIN may be abbreviated to S, NS, C48, C60, and SM,
                 respectively.

Return Codes: 0 - Successful return.
                 4 - Syntax errors detected.
                 8 - Insufficient memory.

Description: *PL1SCAN scans a PL/I source program for syntax errors.
                 However, it cannot detect all errors, because it only
                 scans one statement at a time and does not check
                 interstatement dependencies (i.e., DECLARE statements,
                 undefined labels, etc.).

If SPUNCH is specified, the PL/I scanner will write the source program to the file specified. This is especially useful if SCARDS is assigned to the terminal (*SOURCE*).

Comments: It is often advantageous to run *PL1SCAN since it uses considerably less CPU time and virtual memory than the PL/I compiler.

When a listing of the source code is obtained, each source line is preceded by its line number (in a style similar to MTS).

The error messages are of the form

    IKMxxx line# error-message-text

where "line#" is the line number of the offending card image.

January 1987

*PL1TIDY

Contents: The PL/I "tidy" program.

Purpose: To edit PL/I source programs into an easily readable
                 format.

Use: The program is invoked by the $RUN command.

Program Key: *PL1TIDY

Alt. Name: *PL/1TIDY

Logical I/O Units Referenced:
                 SCARDS - the source program to be edited.
                 SPRINT - a listing of the edited source program and
                          diagnostics.
                 SPUNCH - the edited source program.
                 SERCOM - error messages plus a message for each external
                          procedure.

Return Codes: 0 - Successful return.
                 4 - Warning messages printed.
                 8 - Errors detected.
                 12 - Errors detected.
                 16 - Fatal errors detected.

Description: *PL1TIDY edits PL/I source programs into an easily
                 readable form. This feature is especially desirable to
                 clean up files containing modifications made from termi-
                 nals. The program will indent each statement by an
                 amount corresponding to its nesting depth. This program
                 can also be used to edit programs into an acceptable form
                 which cannot be compiled because of the rather stringent
                 SORMGIN constraints of *PL1 and *PLC. Programs also can
                 be edited into "compressed" form with all nonsignificant
                 blanks removed, thus minimizing the size requirements of
                 the file containing the source program.

                 The complete description of *PL1TIDY is given in MTS
                 Volume 7, PL/I in MTS.

<u>*PRODUCTSUPPORT</u>

Contents: A list of the hardware and software products supported by
                the Computing Center and other University units.

Use: The list may be obtained via the $COPY command, e.g.,

                    $COPY *PRODUCTSUPPORT

364.2 *PRODUCTSUPPORT

January 1987

<u>*PROFORT</u>

Contents: The FORTRAN Execution Profiler.

Use: The program is invoked by the $RUN command.

Program Key: *PROFORT

Return Codes: Always zero.

Description: The Profiler is a tool for analyzing the performance of
                 FORTRAN programs. The Profiler first runs a FORTRAN
                 program, then produces a source code listing of the
                 program showing how many times each source statement was
                 executed during the run and how many times each path of
                 the program was traversed.

                 One strategy for testing programs is to feed the program
                 a sufficient variety of test data (not necessarily in the
                 same test run) to cause every possible path to be
                 traversed at least once.   This does not guarantee a
                 bug-free program, but at least it leaves no code entirely
                 untested.   A profiler is a useful tool for applying this
                 strategy. One looks for zero counts and creates addi-
                 tional test data to eliminate them.

                 A profiler is also useful for exposing bugs, by virtue of
                 its power to reveal erroneous loop counts, excursions
                 into inappropriate portions of code, and other unexpected
                 behavior.

                 The information provided by a profiler forms a good basis
                 for analyzing algorithms and their implementations in
                 code. The most frequently executed portions of a program
                 are not necessarily where the most time is spent, but
                 they often are. Consequently, high execution frequencies
                 are a good indicator of where program bottlenecks exist.

                 Donald Knuth, together with some colleagues and students,
                 studied the behavior of FORTRAN programs, using several
                 different methods of analysis. He concludes (<u>Software--
                 Practice and Experience</u>, 1:105-133, 1971): "The program
                 profiles (i.e., collections of frequency counts) which we
                 used in our analyses turned out to be so helpful that we
                 believe profiles should be made available routinely to
                 all programmers by all of the principal software sys-
                 tems."   Also, "...our group came to the almost unanimous
                 conclusion that all software systems should provide
                 frequency counts to all programmers, unless specifically
                 told not to do so."

Richard Sites (<u>ACM Sigplan Notices</u>, 13:12:98-101, December 1978) goes so far as to say, "Statement counting is the single most useful tool that a programming system can provide to the user."

For the complete description of the Profiler, see MTS Volume 6, <u>FORTRAN in MTS</u>.

*PROLOGC, *PROLOGW

Contents: Driver programs for interpreters for the Prolog language.

Use: The driver program is invoked by the command

                    $RUN *PROLOGC

                or

                    $RUN *PROLOGW

Program Key: PLOG:PROLOG

Logical I/O Units Referenced:
                None

Description: These files contain the driver programs for two interpre-
                ters for the Prolog language.

                The program in *PROLOGC implicitly invokes VSS and Prolog
                runs under VSS using CMS simulation. The syntax is based
                on the Edinburgh dialect of the Prolog syntax that is
                used by Clocksin and Mellish in their book, _Programming_
                in Prolog. While there is no official standard available
                for Prolog, the Clocksin and Mellish sytax is more or
                less accepted as being standard.

                The program in *PROLOGW implicitly invokes VSS and Prolog
                runs under VSS using CMS simulation. The syntax is based
                on the Waterloo syntax which varies from the syntax used
                by Clocksin and Mellish in their book, _Programming in_
                Prolog. This version is made available for compatibility
                with the older version of PROLOG that was available in
                PLOG:PROLOG.

                The programs are best suited for use in interactive mode.
                They do not use the usual MTS I/O units, SCARDS, SPUNCH,
                etc.   Input is entered at the terminal or from a file
                (using the consult predicate provided in the language) or
                by combining both. The program output can be directed to
                a file. The diagnostic messages are written to the
                terminal.

                For details on running the interpreters, see CCMemo 478,
                "Prolog on MTS - Core Sntax," and CCMemo 477, "Prolog on
                MTS - Waterloo Syntax."

366.2 *PROLOGC, *PROLOGW

January 1987

<u>*RATES</u>

Contents: The current rate schedule for the use of the computer.

Use: The rate schedule may be obtained via the $COPY command,
               e.g.,

                   $COPY *RATES

January 1987

<u>*RATFOR</u>

Contents: A FORTRAN preprocessor program.

Use: To allow control structures for FORTRAN programs.

Program Key: *RATFOR

Logical I/O Units Referenced:
                SCARDS - structured FORTRAN input.
                SPRINT - structured program listing.
                SPUNCH - processed standard FORTRAN source output.

Return Codes: 0 - Successful return.
                4 - Syntax errors detected.

Description: RATFOR (Rational FORTRAN) is a FORTRAN language prepro-
                cessor developed by Kernighan and Plauger and described
                in the publication

                      <u>Software Tools</u>, by Brian W. Kernighan and P. J.
                         Plauger, Addison-Wesley, 1976

                RATFOR was developed for the purpose of overlaying more
                modern and commonly accepted control structures and
                features on FORTRAN, making it a more palatable and
                versatile tool to utilize in the solution of programming
                tasks that for one reason or another require the use of
                FORTRAN. The language description as well as the prepro-
                cessor itself are described in the book and the prepro-
                cessor as implemented on MTS is very straight forward to
                use.

                The structured FORTRAN program is read by the translator
                from logical I/O unit SCARDS and the standard FORTRAN
                source code is produced on logical I/O unit SPUNCH.   A
                listing is produced on logical I/O unit SPRINT in batch
                mode and also in terminal mode if SPRINT is explicitly
                assigned to a file or device. The resulting standard
                FORTRAN output is ready for processing by a standard
                FORTRAN compiler.

Example: $RUN *RATFOR SCARDS=RATPROG SPUNCH=FORTPROG

                      In the above example, the structured FORTRAN program
                      in the file RATPROG is processed into a standard
                      FORTRAN program and written into the file FORTPROG.

370 *RATFOR

January 1987

*REDUCE

Contents: The REDUCE algebraic language (version 3.2).

Use: The program is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:
         SCARDS - default input stream for REDUCE commands.
         SPRINT - default output stream for REDUCE output.
         GUSER - input stream after an attention interrupt when
                  SCARDS is not assigned to the terminal.
         SERCOM - output stream after an attention interrupt when
                  SPRINT is not assigned to the terminal.
         1   - REDUCE checkpoint image used for the automatic
                  restore facility.
         10   - file to use for FAPLIB.
         11   - file to use for XMPLIB.

Parameters: The following parameters may be specified in the PAR
               field of the $RUN command.

         BIG   The BIG parameter invokes a version of REDUCE that
                provides extra space for BPS (45000 instead of the
                normal 18000).   This should be specified if the
                symbolic integrator or other large parts of REDUCE
                are to be used.

         RLISP The RLISP parameter invokes RLISP instead of
                REDUCE.

         SLISP The SLISP parameter invokes SLISP instead of
                REDUCE.

Return Codes: 0 - Successful return.
               8 - Out of memory or garbage collection failure.
               12 - Internal error (should not occur).

Description: REDUCE is a LISP program designed for general algebraic
               computations including those of interest to physicists
               and engineers. Its capabilities include:

         (1) expansion and ordering of polynomials and ration-
              al functions,
         (2) substitutions and pattern matching in a wide
              variety of forms,
         (3) automatic and user controlled simplification of
              expressions,
         (4) calculations with symbolic matrices,

        (5) arbitrary precision integer and real arithmetic,
        (6) facilities for defining new functions and extend-
             ing program syntax,
        (7) analytic differentiation and integration,
        (8) factorization of polynomials, and
        (9) Dirac matrix calculations of interest to high
             energy physicists.

The file *REDUCE contains a small program that sets
certain default I/O assignments and then calls standard
LISP to restore the REDUCE checkpoint image and execute
REDUCE. The usual way to run REDUCE from a terminal or
batch job is

        $RUN *REDUCE
        <REDUCE commands>
           .
           .
           .
        BYE;

For further information on REDUCE, see the <u>Reduce User's
Manual</u> by Anthony C. Hearn and the <u>Reduce User's Guide
for IBM 360 and Deriviative Computers</u> by Larry R. Seward.

January 1987

*RESTORE

Contents: The system file restore program.

Purpose: To restore files saved by the system file-save procedure.
               The Computing Center runs a special system program every
               day which saves all user files that have changed since
               the last file-save. This is done to enable files to be
               recovered in the event of major disk failures. It is
               possible for users to take advantage of this procedure to
               restore files to their status as of any day in the recent
               past (about six weeks). Files to be restored must be
               owned by the signon ID running the program.

               Note: This program has no relation to *FS which users
               may use both to save and restore files on their own
               magnetic tapes.

Use: The program is invoked by the $RUN command. When
               execution begins, *RESTORE expects the user to enter a
               command that indicates which of several services is
               desired. For example, the DISPLAY command may be used to
               determine which versions of a file are available for
               restoration. The RESTORE command may be used to indicate
               the name of a file to be restored. Several successive
               RESTORE commands may be entered; the file names are
               queued and then the files are restored after the QPROCESS
               or STOP command is given. The queue may be emptied with
               the CANCEL command.

               After a QPROCESS, STOP, or RETURN command or an end-of-
               file is entered, the program automatically requests the
               operator to mount the proper tape(s) and searches them
               for the proper files. If a tape drive is unavailable or
               if the desired tape is in use, a message to the user will
               indicate which files were not restored. Since tape
               mounting and searching are somewhat time consuming, the
               restore operation may take several minutes. It may be
               aborted by an attention interrupt.

               If the file to be restored already exists, confirmation
               will be required to destroy the existing version of the
               file. The user may respond with "OK", "YES", or "!". If
               the program is run in batch mode, confirmation is not
               required and the current version of the file will be
               replaced by the restored version.

               In addition to restoring the data of the file, *RESTORE
               will restore the permit access information for the file.

Comments: *RESTORE does not check to determine whether files queued
                for restoration will fit in the disk space allocated to
                the signed on ID. Restore operations are aborted if the
                user's file space limit is exceeded while a file is being
                restored. The save tapes used by *RESTORE only contain
                files which have existed during the most recent past
                (about six weeks). However, the Computing Center main-
                tains a separate set of tapes to provide file restoring
                service for older files that are not available through
                *RESTORE. A request must be made to the Computing Center
                Business Office, if a user requires an older version of a
                file (one that is not contained in the tapes used by
                *RESTORE),

                If *RESTORE is run with a global or local time estimate,
                the estimate should be at least 20 seconds per file in
                order to insure a reasonable probability of success in
                all cases.   The approximate cost for restoring one file
                is $3.00-$5.00, but may be as high as $10.00 or more,
                depending on the file size and its position on the tape.

                For extremely large files, *RESTORE will buffer the file
                on disk during the restoration process. As a conse-
                quence, the tape may not be referenced for a long period;
                hence, users may get a tape-timer message, but this
                should not be cause for alarm.

Program Key: *RESTORE

Logical I/O Units Referenced:
                SCARDS - the source of commands and responses to prompt-
                         ing messages (defaults to *SOURCE*).
                SPRINT - prompting messages and listings of files and the
                         times they were saved (defaults to *SINK*).
                SERCOM - error messages. SERCOM is reassigned to *MSINK*
                         when the files are being restored.
                GUSER - response for requested confirmation (OK) to
                         destroy the current version of a file so that it
                         may be replaced with an older version.

Parameters: Any single command may be specified in the PAR field of
                the $RUN command.

Commands: The following commands are available for *RESTORE.   The
                underlined portion of each command may be used as an
                abbreviation.

                CANCEL [filename]

                    If no parameter is given, all files queued for
                    restoration by all previous RESTORE commands are
                    removed. If "filename" is given, the specified

374 *RESTORE

file(s) is removed. "filename" may be a patterned
file name, e.g., PROG? or ?DATA?.

DISPLAY filename [mm/dd/yy] [hh:mm:ss] [HISTORY={n|ALL}]

If "filename" is specified, then all of the versions
of "filename" available for restoring are listed.
If the date or the date and time are given, only
versions on or before that date are   listed.
"filename" may be a patterned file name, e.g.,
PROG?, ?DATA?, or ?.

The HISTORY option specifies the number of old
versions to be displayed. The default is HISTORY=
ALL for nonpatterned file names and HISTORY=1 for
patterned file names.

HELP [COMMANDS]

If only the HELP command is given, an overview of
*RESTORE is given. If the COMMANDS parameter is
specified, a listing of *RESTORE commands is given.
This information is given in full-screen format for
terminals that support full-screen mode.

MCMD command
$command

The input line is executed as an MTS command. After
completion of the command, control is returned to
*RESTORE.

MTS

Control is returned to MTS command mode.   Execution
of *RESTORE may be resumed by issuing the $RESTART
command.

NOQUIT

In batch mode, the program will sign off the job if
any of the files could not be restored. If NOQUIT
is specified, the job will not be signed off.

QDISPLAY

This command displays the names of all files queued
for restoration.

QPROCESS

This command processes all queued restore requests.
If any requests cannot be processed, due to problems

in acquiring a tape drive or restoring the file, they are left in the queue. This permits the user to retry the restore requests at a later time.

RESTORE filename [mm/dd/yy] [hh:mm:ss] [NEWNAME=filename]
      [REPLACE={YES|OK|NO|PROMPT}]

If only "filename" is specified, the most recent version available of that file is queued for restoration. The file will be restored with the same name unless a NEWNAME is given. If a file having the same name currently exists, a warning is printed since restoration will destroy it. In conversational mode, the user will be required to answer when asked if the file should be queued for restoration. Confirmation to allow destroying an existing file will be required when the file is actually being restored. "OK" may be entered for confirmation. The file must be owned by the signon ID running the program, and the user must have "destroy" access to the file. "filename" may be a patterned file name, e.g., PROG?, ?DATA?, or ? (patterned file names may not be used with the NEWNAME option).

If a date (or date and time) is specified, the most recent version saved on or before the date or date/time specified is queued for restoration. When used in conjunction with the file-name pattern "?", this enables a user to restore all files under a signon ID to a version not later than the time and date specified. It should be remembered that since files are saved only when they have been changed, specifying a date actually amounts to requesting that a file be restored to its status as of that date.

The REPLACE option controls whether currently existing files queued for restoration are replaced. If YES or OK is specified, existing files will be replaced. If NO is specified, existing files will not be replaced. If PROMPT is specified, the user will prompted whether to replace existing files. In all cases, non-existing file will be replaced.

RETURN
STOP
<end-of-file>

Terminates command input, attempts to restore all files queued for restoration, and terminates execution of the program.

January 1987

                <attention>

                    If STOP or QPROCESS has not been given yet, an
                    attention interrupt terminates the current command
                    and requests another. If STOP or QPROCESS has been
                    given (while the program is in the process of
                    restoring files), no more files are restored, a list
                    of unrestored files is printed, and the program
                    terminates (STOP) or returns to command mode
                    (QPROCESS).

Return Codes: 0 - Successful return.
               4 - Internal error in program.

Example: The following is a sample terminal session using *RESTORE
              (user input shown in lowercase):

```
#run *restore
#Execution begins

 Type command.
*? dis callmacros

     1 CALLMACROS saved SUN 10/28/84 09:23:02
     2     "      saved TUE 10/23/84 02:59:08
     3     "      saved SUN 10/21/84 09:40:34
     4     "      saved SAT 10/20/84 02:56:33

 Type command.
*? res  callmacros 10/28/84
 CALLMACROS (SUN 10/28/84 09:23:02) has been queued for restoration.

 Type command.
*? qprocess
 It will take some time to mount and scan the file save tape.
 Use ATTN to abort.
 W999:CALLMACROS 09:23:02 10-28-84 SUNDAY
 "CALLMACROS" is to be replaced with a new version. OK?
*? ok
 Completed
 T905 released.
 Type command.
*? dis d?

     1 DG saved TUE 10/30/84 03:17:01
     1 DIRECTORY00 saved SUN 10/28/84 09:26:27
     1 DISASTER saved SUN 10/28/84 10:56:36
     (There are 5 older versions of this file.)

 Type command.
*? dis disaster history=5

     1 DISASTER saved SUN 10/28/84 10:56:36
```

```
       2       "     saved SUN 10/21/84 10:06:44
       3       "     saved SUN 10/14/84 10:12:13
       4       "     saved SUN 10/07/84 10:21:37
       5       "     saved MON 10/01/84 01:41:28
     (There is 1 older version of this file.)

 Type command.
*? stop
 Total files restored: 1
#Execution terminated
```

January 1987

<u>*RG</u>

Contents: The Report Generator program.

Use: To generate reports from a database.

Program Key: *RG

Logical I/O Units Referenced:
                SCARDS - specifies the RG commands and the master file.
                         The master file can also be assigned by the
                         DATAFILE=FDname parameter (see below), in which
                         case only the RG commands are read from SCARDS.
                SPRINT - specifies the output device used to print the
                         report.
                SERCOM - specifies the output device used to print the
                         command listing (if requested), error comments,
                         and execution-time statistics.
                0   - specifies the macro library   to   be   used
                         (optional).

Parameters: The following parameters may be specified on the PAR
                field of the $RUN command.   The parameters must be
                separated by commas.

                LIST/NOLIST   The LIST parameter lists the RG commands
                                 used for the run including the line
                                 numbers. The default is NOLIST in ter-
                                 minal mode, and LIST in batch mode.

                LINESIZE=n   The LINESIZE parameter sets the width of
                                 the output reports to "n" characters.
                                 The default is 132 characters, which
                                 includes printer carriage control. The
                                 maximum width that may be assigned to
                                 LINESIZE is the maximum output record
                                 length of the file or device that is
                                 assigned to SPRINT.

                DATAFILE=FDname The DATAFILE parameter changes the de-
                                 fault input FDname for the master file.
                                 Normally, the master file is assigned to
                                 SCARDS.   The FDname <u>must</u> be terminated
                                 by a blank.

                ITEMS=n   The ITEMS parameter requests that only
                                 the first "n" items are to be read from
                                 the master file. The default is all
                                 items.

Return Codes: None.

Description: The Report Generator is a simple information-retrieval
                system which reads a data file once (sequentially), and
                produces one or more reports based on the contents of
                that file. This master file may have almost any format
                as long as it is consistent and definable.

                RG has the capability of extracting some or all of the
                data from a file and printing the extracted data in a
                user-defined format.   RG is also capable of producing
                KeyWord-In-Context (KWIC) indices.

                RG also may be used to generate a list of the different
                values contained in a field and a count of the number of
                times each value occurs. For example, a mailing list may
                be scanned to list all the different cities in the file
                and the number of occurrences of each.

                RG has only limited arithmetic capabilities.   It is
                possible to generate one field from another using the
                simple operators plus(+), minus(-), multiply(*), and
                divide(/). However, it is not possible to perform any
                computation from record to record on any field.

                A run of RG is divided into two parts which provide the
                following information:

                    (1) A description of the master file format.
                    (2) A description of the report format.

                The first part consists of three commands: the ITEM,
                ENTRY, and FIELD commands. The ITEM command specifies
                which physical records belong to one item or logical
                record.   The ENTRY command specifies particular physical
                records within the item. The FIELD command specifies the
                location of each field in the physical record. A fourth
                command, the END command, terminates the first part.

                The second part consists of five commands: the EXTRACT,
                TABLE, ORDER, TITLE, and PRINT commands.   The EXTRACT
                command specifies which items are to be used in the final
                report.   The ORDER command orders the extracted items.
                The TITLE command assigns title(s) to the report, and the
                PRINT command describes the actual report format.   The
                TABLE command generates frequency tables of occurrences.
                The END command also terminates the second part.

                The complete description of the Report Generator program
                and its command language is given in Computing Center
                Memo 437, "UBC RG: The Report Generator."

*RMLOG

Contents: A program to scan the Resource Manager log files.

Use: The program is invoked by the $RUN command.

Program Key: *RMLOG

Logical I/O Units Referenced:
                INPUT - input commands
                SERCOM - error messages

Return Codes: None

Description: *RMLOG contains a program for reporting extracts from the
                Resource Manager log files.   These log files contain
                recent history data from the spooling system. Currently
                the Resource Manager keeps 8 days worth of history data.
                This data is currently updated only hourly, so *RMLOG may
                not have data for the period from an hour ago to the
                current time. The Resource Manager log files contain
                data on spooled jobs such as print and batch jobs, and on
                BITNET jobs such as file transfers and messages.

                The program is invoked with the command

                     $RUN *RMLOG [INPUT=input] [SERCOM=error-messages]

                The input read from INPUT (SCARDS) consists of an
                arbitrary number of commands. A command is one of the
                following:

                    $MTS-command   Issues the given MTS command.

                    *comment   Does nothing.

                    FORMAT {FREE|FIXED}
                                     Sets the format for printed output.

                    HELP topic   Prints help text for the given topic.

                    MTS   Returns to MTS, can be restarted.

                    PRINT options   Selects and prints entries from the
                                     RM log files. See the description of
                                     the print options below.

                    STOP   Stops the program.

| TIME {OFF|SHORT|DATE|FULL}
| Sets the time format
|
| PRINT Command Options
|
| The PRINT command has many options for selecting the log
| entries to be printed. The only one of these that limits
| the amount of data the program searches through is the
| TIME option. This is because the log files are ordered
| by time and have no indexes. The program sequentially
| reads each log file that could have data from the
| requested time. Currently each log file contains a day's
| data, with the data starting at about 8:00 AM each day.
| Thus for example, a request to print data from 9:00 AM to
| midnight of a particular day will search only one file,
| whereas a request with no time specified will search all
| log files.
|
| More than one print option can be specified by separating
| them with blanks in the command. All of the options must
| match a record in order for the record to be selected for
| printing. The print options, with the minimum abbrevia-
| tions underlined, are:
|
| DESTNODE=dest-node   Select BITNET entries with the
| given destination node.
|
| DESTUSER=dest-user   Select BITNET entries with the
| given destination user.
|
| DEVICE=device-name   Select entries with the given
| device name.
|
| ID=origin-id   Select entries with the given
| four-digit origin-id.
|
| JOB#=job-number   Select entries for the given
| job number.
|
| JOBNAME=job-name   Select entries for given
| job-name.
|
| KIND=record-types   Select entries of the given
| record-types. See the list of
| record types below.
|
| LINK=link-name   Select BITNET entries with the
| given link-name.
|
| MESSAGE=msg-number   Select entries with the given
| message number.

MSG#=msg-number	Select entries with the given message number.
ON FDname	Print the entries on the given FDname.
ORIGID=origin-id	Select BITNET entries with the given four-digit origin-id.
ORIGNODE=origin-node	Select BITNET entries with the given origin-node.
PROCESSOR=proc-name	Select entries for the given processor.
QUEUE=queue-name	Select entries for the given queue.
ROUTE=route	Select entries with the given print route.
SENTNODE=sent-node	Select BITNET "sent" messages from the given node.
SENTLINK=sent-link	Select BITNET "sent" messages for the given link.
TIME="time/date-range"	Select entries within the given time/date-range. See below for further information on specifying time/date-ranges.
TYPE=record-types	Select entries of the given record-types. See the list of record types below.

Note that you can only access the records for your MTS userID, and that to access the BITNET records for a job, you must specify ORIGID=origin-id to select the records for only that BITNET job. The origin id of a job is the last four digits of the job's Resource Manager job number. Thus if you want to trace the job 119353, you must specify ORIGID=9353. See also the example below detailing the tracing of a BITNET message.

Time/Date Ranges

A time/date range for the PRINT TIME command may be given in the form

        time/date

| or

|

| time/date1 TO time/date2

|

| Time/date input conversion is driven by a grammar that
| attempts to accept most styles of time/date presentation
| that are commonly used. It generally requires that the
| components specified be meaningful and unambiguous.

|

| An input time may be composed of the following pieces.
| They may be put together in most meaningful combinations,
| using commas and blanks as conventional separators.

|

| Year   A year is a two-digit number in the
| range 32 to 99, inclusive, or a four-
| digit number in the range 1900 to 1999,
| inclusive. In some unambiguous con-
| texts, a two-digit number in the range
| 0 to 31 will also be recognized as a
| year.

|

| Month   A month is one of "January",
| "February", ..., "December". Month
| names may be abbreviated to their first
| three characters.

|

| Day   The day of the month is specified by a
| number in the range 1 to 31, inclusive.

|

| Time   Times are integer strings containing
| ":" and/or "." and/or followed by
| "AM" or "PM" (or "A.M." or "P.M.") A
| fully specified time appears as
| "23:22:05.67835".

|

| The time must specify the hours and
| minutes, with the seconds and micro-
| seconds components optional. The
| minutes component may also be omitted
| if "AM" or "PM" appears.

|

| Thus, the following are all allowed:

|

| 12 AM
| 23:59
| 00:01:35.5

|

| Day of Week Days of the week are specified as "Mon-
| day", "Tuesday", ..., "Sunday". They
| may be abbreviated to the first three
| characters of the day name ("Mon",
| "Tue", etc.).

380.4 *RMLOG

The day-name is just checked for correctness with respect to the other date components. Currently, a day-name cannot be used by itself.

MM-DD-YY   For compatibility with older date input and output routines the MM-DD-YY form of month, day and year is supported (it may also be specified as MM/DD/YY). For example, "11-06-88" is November 6, 1988.

This form is not recommended because of its ambiguity. Depending on where you live, the convention may be MM-DD-YY, DD-MM-YY, or YY-MM-DD. This package supports the MM-DD-YY form because that form has been the MTS standard.

IBM   For compatibility with certain IBM software, the form YY.DDD is supported, where YY is a two- or four-digit year number and DDD is the day number in the year. For example, "89.103" is April 13, 1989.

Miscellaneous Forms
                The following strings are recognized, with the obvious meanings.

                        YESTERDAY
                        TODAY
                        NOW

These first two forms represent a date only, and may be combined with a time specification. The form "NOW" repre- sents the current time (to the micro- second level).

It is not necessary to specify all the components of the time (year through microsecond). The input conversion routines will default unspecified components in many cases.

When converting, the default is selected with respect to the current time. Any unspecified low-order components are always set to the beginning of the interval. Unspe- cified high-order components are copied from the current time.   For example, if it is currently any time in 1990, then "Jan 25" is equivalent to "00:00:00.0 Jan 25/90".

| The following examples illustrate the use of time/date
ranges:

       PRINT TIME="Jan 18/1990"
       PRINT TIME="Thu January 18/90"
       PRINT TIME="Jan 18"
       PRINT TIME="1/18/90
       PRINT TIME="1-18-90"
       PRINT TIME="90.018"
       PRINT TIME="YESTERDAY"
       PRINT TIME="Jan 18/90 To Jan 19/90"
       PRINT TIME="January 18/1990 To NOW"
       PRINT TIME="11:00 Jan 18/90 To 18:00 Jan 18/90"
       PRINT TIME="6:00 PM YESTERDAY To 6:00 AM TODAY"
       PRINT TIME="11:15:20 Jan 18 To 11:45:30 Jan 18"
       PRINT TIME="January"
       PRINT TIME="January 1990"

Record Types

The record types can be either a single record type, or a
parenthesized list of record types, separated by blanks
or commas. The individual record types are:

    QUEUE   A record of a job being put on a queue.

    PROC   A record of a job being given to a
                processor.

    ACC   A record of the accounting data from the
                completion of a job.

    DONE   A record of the completion of a job.

    NETJOBIN   A record of the receipt of a BITNET job.

    NETJOBOUT A record of the sending of a BITNET job.

    NETMSG   A record of a BITNET network message or
                command which is not a message saying that
                a job from this site has been relayed
                onwards.

    MSGJOBIN   A record of the transformation of an
                incoming BITNET message to a message-
                system message.

    MSGJOBOUT A record of the transformation of an out-
                going BITNET message from a message-system
                message.

380.6 *RMLOG

| RNMSENT   A record of a BITNET network message that
| indicates that an intermediary site has
| forwarded a job from this site onwards.
|
| Additionally, the following aggregate abbreviations are
| defined:
|
| JOB   means (QUEUE, PROC, DONE, ACC)
|
| NETJOB   means (NETJOBIN, NETJOBOUT, RNMSENT)
|
| MSGJOB   means (MSGJOBIN, MSGJOBOUT)
|
| NET   means (NETJOB, MSGJOB)
|
| SENT   means (NETJOBOUT, RNMSENT)
|
| Examples: The following example shows the basic use of the program
| to obtain data on jobs that were printed yesterday.   The
| "Log__" is a prompt printed by the program; the remainder
| of the line is typed by the user.
|
| # run *rmlog
| Log__ print time="yesterday"
| Wednesday January 10, 1990
| 16:41 1ABC:RM193730 queued to CNTR.
| 16:43 1ABC:RM193733 queued to UNYN.
| 16:43 1ABC:RM193733 on proc PTR2.
| 16:43 1ABC:RM193733 on UNYN PTR2, 61 Ppp, 8,378 Ppl,
| 61 PPI, 31 PPS.
| 16:43 1ABC:RM193733 done.
| 16:45 1ABC:RM193730 done.

380.8 *RMLOG

January 1987

*SDL

Contents: The Survey Definition Language.

Use: SDL is invoked by the $RUN command.

Program Key: *SDL

Logical I/O Units Referenced:
                 SCARDS - survey description statements.
                 SPUNCH - resulting survey object program.

Return Codes: 0 - Successful return.

Description: SDL is a specialized programming language consisting of a
                 small number of straightforward statements which can be
                 used to write programs that send a sequence of requests
                 for information interactively to an individual at a
                 terminal. Replies to the program's prompts are read
                 immediately and can be verified at each step, stored for
                 later analysis, or used to branch to appropriate subsec-
                 tions of a question list.

                 The most common application of SDL is the development of
                 computerized survey questionaires. Another potential
                 application is data entry, either in conjunction with a
                 survey, or as a way to enter previously selected or coded
                 data.

                 The Survey Definition Language is completely described in
                 Computing Center Memo 452, "Survey Definition Language:
                 *SDL".

January 1987

*SIDEBYSIDE

Contents: A multiple-page output processing program.

Purpose: To place consecutive pages side-by-side on a single page,
                 taking logical carriage controls into consideration.

Use: The program is invoked by the $RUN command.

Program Key: *SIDEBYSIDE

Logical I/O Units Referenced:
                 SCARDS - input data which is to be placed side-by-side on
                          an output page.
                 SPRINT - output containing the input data after it has
                          been placed in a side-by-side format.
                 SERCOM - error messages.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command.   The underlined portion of
                 each parameter may be used as a minimal abbreviation.

                 INLINES=n   "n" specifies the maximum number of
                                  actual input lines for a page-column in
                                  the output.   Overprinting lines (those
                                  with a "+" carriage control) should each
                                  be counted as one line. The default is
                                  INLINES=132 lines.

                 WIDTH=n   "n" specifies the width of each column
                                  on an output page. The specified width
                                  should be large enough to include any
                                  necessary additional space on the right
                                  to produce blank space between the col-
                                  umns (gutters). Input lines longer than
                                  this width will be truncated.   The de-
                                  fault is WIDTH=66 print positions.

                 COLUMNS=n   "n" specifies the number of columns, or
                                  input pages, that will be placed side by
                                  side on an output page. The default is
                                  COLUMNS=2.

                 TOPMARGIN=n   "n" specifies the number of physical
                                  lines before the first logical line on
                                  the page. This parameter describes the
                                  printer layout that will be used to
                                  print the output and should reflect the
                                  characteristics of the printer and type

style being used. The default is
TOPMARGIN=3 lines.

LOGICALLINES=n   "n" specifies the number of print lines
                 per logical page. This parameter de-
                 scribes the printer layout that will be
                 used to print the output and should
                 reflect the characteristics of the
                 printer and type style being used. The
                 default is LOGICALLINES=60 lines.

PHYSICALLINES=n "n" specifies the number of print lines
                 per physical page. This parameter de-
                 scribes the printer layout that will be
                 used to print the output and should
                 reflect the characteristics of the
                 printer and type style being used. The
                 default is PHYSICALLINES=(LOGICALLINES +
                 2*TOPMARGIN) lines.

OUTLENGTH=n   "n" specifies the maximum number of
                 characters, including carriage control,
                 to be placed in each output line. Char-
                 acters beyond the maximum output length
                 are discarded.   The default is OUTLEN=
                 columns*width+1 characters.

NO6OR8   If NO6OR8 is specified, it is assumed
                 that logical carriage controls "6" and
                 "8" are not recognized as such by the
                 output device. In this case, the effect
                 of any such carriage controls in the
                 input is simulated in the output by the
                 program. It is not necessary to specify
                 this parameter if the input does not
                 contain logical carriage controls "6" or
                 "8." (This parameter is relevant only
                 for output printed at installations,
                 such as Wayne State University, that do
                 not recognize the "6" and "8" carriage
                 controls.)

Return Codes: None.

Description: The program reads the input from SCARDS and divides it
               into print pages. Groups of consecutive pages, with the
               number of pages per group equal to the COLUMNS parameter,
               are then placed in a side-by-side format, and each group
               is written as a single print page to SPRINT.

```
Examples:     $RUN *SIDEBYSIDE SPRINT=-OUT PAR=W=12,C=4,NO6OR8
              1ONE
              8EIGHT
              2TWO
              2TWO
              <LESS THAN
              4FOUR
              4FOUR
              4FOUR
              4FOUR
              ;SEMICOLON
              6SIX
              6SIX
              6SIX
              6SIX
              6SIX
              6SIX
```

         In the above example, the following records would be
         placed in the file -OUT:

```
;                                 SEMICOLON
1ONE          TWO         FOUR    SIX
-
-
-
                                  SIX
4                         FOUR
-
0                                 SIX
2             TWO         FOUR     SIX
-
-
-
                                  SIX
4                         FOUR
-
0                                 SIX
-
-
-EIGHT
<             LESS THAN
```

If the NO6OR8 parameter were omitted and COLUMNS were set to 3, -OUT would contain:

```
1ONE          TWO          FOUR
4                          FOUR
2             TWO          FOUR
4                          FOUR
8EIGHT
<             LESS THAN
9SEMICOLON
1SIX
6SIX
6SIX
2SIX
6SIX
6SIX
```

The following two sheets contain sample output using *SIDEBYSIDE.

The first sheet uses *SIDEBYSIDE in conjunction with *PAGEPR (FORMAT=TWOUP) to produce a "FOURUP" form, in which eight pages fit on one sheet. The text used is the first eight pages of the file *CCPUBLICATIONS. The command to produce this were:

```
$RUN *SIDEBYSIDE SCARDS=-CCPUB SPRINT=-P PAR=W=72
$RUN *PAGEPR SCARDS=-P PAR=FORMAT=TWOUP,MARGIN=.1
```

The second sheet uses *SIDEBYSIDE in conjunction with *PAGEPR (FORMAT=LANDSCAPE) to produce two side-by-side pages on each side of a sheet. The text used is the first four pages of the file *CCPUBLICATIONS. The command to produce this were:

```
$RUN *SIDEBYSIDE SCARDS=-CCPUB SPRINT=-P PAR=W=72
$RUN *PAGEPR SCARDS=-P PAR=MARGIN=.2
```

390 *SIDEBYSIDE

<u>*SIGSETUP</u>

Contents: A program for executing selected commands from a sigfile.

Program Key: *SIGSETUP

Logical I/O Units Referenced:
                SCARDS - input lines to the setup program.
                SPRINT - output from the setup program.
                GUSER - input for the Read Expression operand.

Return Codes: 0 - Successful return.
                4 - Invalid input line encountered.

Description: *SIGSETUP contains a program for use in sigfiles to
                selectively execute various commands.

                The program reads input lines consisting of expressions
                and commands to be performed if the expression is true.
                The expressions can test various conditions such as the
                type of device at which the user has signed on, the day
                of the week, or the userID signed on. The commands can
                be used to set MTS or device-support options, dynamically
                load and run other programs, print messages, etc. Vari-
                ous items can be substituted into a command before it is
                executed.

                The program is invoked (usually from a sigfile) with a
                command of the form

                    $RUN *SIGSETUP SCARDS=input SPRINT=output PAR=ECHO

                The input read from SCARDS consists of an arbitrary
                number of lines of the form

                    expression command-list

                where "expression" is as described below. The expression
                is evaluated, and if the result is true, then all
                commands in "command-list" are performed. If it is
                false, all commands are skipped.

                "command-list" consists of zero or more commands as
                described below. Commands are separated by slashes (/).
                If the character / is required as part of a command, it
                should be replaced by //.

                The values of certain "variables" may be substituted into
                a command before it is performed. An item to be
                substituted is delimited with a vertical bar (|) on each

                                                      *SIGSETUP   390.1

side.   If a "|" is required as part of a command, it is
represented by "||".

If the ECHO parameter is specified, each command to be
performed is written to SPRINT before it is executed.

An input line that begins with an asterisk (*), or that
is completely blank, is ignored by the program.   Such
lines can be used to annotate the sigfile commands.

Expressions

An "expression" is made up of terms and binary expres-
sions, connected together by the operators &, |, and ~.
Parentheses may be used in expressions to indicate
intended order of evaluation.

A term in an "expression" may be:

BATCH   true if user signed on in batch.

TERMINAL   true if user signed on interactively.

ALL   always true.

DAILY   is true if this is the first time the
                user has signed on for the day.

WEEKLY   is true if the user has not signed on
                since the previous Sunday.

MONTHLY   is true if this is the first time the
                user has signed on in the current month.

YEARLY   is true the first time the user signs on
                for the year.

day of week   is true each time the user signs on
                during the specified day (e.g., MONDAY,
                TUESDAY).

character string
                Any other character string entered as a
                term is compared to the device name and
                type at which the user is signed on.
                Both the local (MTS) name and type and
                the remote (Merit/UMnet network) name and
                type are tested. Thus, the term "xxxx"
                is treated the same as the expression

                    NAME=xxxx | TYPE=xxxx | TERMTYPE=xxxx
                       | TERMNAME=xxxx

390.2 *SIGSETUP

                              See below for an explanation of the NAME,
                              TYPE, TERMTYPE, and TERMNAME variables.

The DAILY, WEEKLY, MONTHLY, and YEARLY items are deter-
mined from the "last signon time" information. For the
purposes of the DAILY and WEEKLY options, the day is
considered to start at 6:00 am, with signons earlier than
that treated as if they were on the previous day. Note
that these options will not be true if the user has
signed on since 6:00 am, even if sigfile processing was
not performed (e.g., because of an attention interrupt).
Similarly, it should be realized that expressions such as

        TERMINAL & DAILY

would be false at each signon of the day if the first is
a batch run. The 6:00 am starting time will usually
allow low-priority batch jobs submitted the previous day
to execute without satisfying the first-signon condition.

A binary expression has the form

        variable operator value

where "variable" is one of the SIGSETUP variables defined
below. "operator" may be any of =, ˜=, <, <=, >, or >=.
A "value" may be a character string, a hexadecimal
string, a number, or a date or time as may be appropriate
to the "variable" specified. Examples of value are given
below.

The allowed variables are:

NAME   specifies the device name of the user's
                MSOURCE device.

TYPE   specifies the device type of the user's
                MSOURCE device. For a Merit/UMnet connec-
                tion, TYPE will be "MNET". For an IBM
                3270 connection, TYPE will be "3270".

TERMTYPE   specifies the remote type of the user's
                MSOURCE device. For a Merit/UMnet connec-
                tion, this is the type by which the
                terminal is known to the network: for an
                IBM PC-compatible terminal, TERMTYPE will
                be "IBMPC"; for a Macintosh Versaterm
                connection, TERMTYPE will be "VERSA"; for
                an Ontel connection, TERMTYPE will be
                "OP/VIS".

TERMNAME   specifies the remote name of the user's
                MSOURCE device. For a Merit/UMnet connec-

                                              *SIGSETUP 390.3

	tion, this contains the Merit address of the terminal port type and the transmission rate. For an IBM 3270 connection, this is the same as NAME.
ID	specifies the userID of the user running the program. When it is used in a binary expression, if the specified value is shorter than four characters, the value will be padded with ".$.", "$.", or "." as necessary to form a valid userID.
	This variable allows using the same sigfile from a number of userIDs, but having some actions performed differently depending on the userID.
PROJECT	is the four-character projectID for the group to which the userID running the program belongs. DEPT and DEPARTMENT are synonyms for PROJECT.
TIME	specifies the _elapsed_ _time_ in milliseconds since execution of SIGSETUP began. With a little tuning, this variable can be used in expressions to skip or perform certain actions when response is bad, or when at a slow-speed terminal.
RAND	results in a new random number in the range 1 to 100 inclusive each time it is used. It may be used in an expression of the form

          RAND < n

          to perform specified commands randomly at about n% of the signons.

DATE	specifies the date when execution of SIGSETUP began.
TOD	specifies the time of day when execution of SIGSETUP began.
SYSTEM	specifies the system being run, generally MTS. This is intended for systems programmers only (for whom it may have the value TMTS or SMTS).
HOST	specifies the hostID of the system on which the program is running. This will

390.4   *SIGSETUP

                         be "UM" for the UM-MTS system, and "UB"
                         for the UB-MTS system.

         GUINFO(guinfoparm)
                         specifies the current value of any item
                         available from the system subroutine GUIN-
                         FO. The parameter guinfoparm may be eith-
                         er an item-name or number (see examples
                         below).

         SENSE(n1:n2)   specifies information returned by the SNS
                         control command for the MSOURCE device.
                         The value is the "n2" characters starting
                         at position "n1" (0-origin).   If   the
                         device does not accept the control com-
                         mand, or if it does not return enough
                         information, the value is a null string.

         READ("prompt") will read a line from the terminal. The
                         input typed becomes the value of the
                         expression. The parameter is optional; if
                         one is given, it will be written to the
                         terminal as a prompt. For example,

                              READ("Set macros on?") = "YES"

                         will issue the prompt and read a response.
                         The response is then compared to the
                         string "YES", and if equal, the expression
                         will be true.

     The values used as the right-hand sides of binary
     expressions may be any of the following:

     number   an unsigned decimal integer.

     mm/dd/yy   a date in the form month/day/year. Either
                      / or - may be used to separate the parts.

     hh:mm:ss   a time-of-day in hours, minutes,   and
                      seconds.

     'hexstring'   a string of hexadecimal digits (0-9, A-F)
                      enclosed in primes ('), with optional
                      interspersed blanks.

     "string"   an arbitrary character string enclosed in
                      quotes ("). A quote within the string may
                      be represented by two quotes.

     other   any sequence of characters which is none
                      of the above, and does not contain certain

                              special characters, will be treated as a
                              character string.

          When dates or times are being compared, they are con-
          verted to Julian values and compared numerically. A
          hexadecimal string may be treated as either a character
          string or a numeric value, as appropriate.

          If a number, date, or time-of-day appears in a context
          requiring a character-string, it will be treated as such.
          Thus, for example,

               TYPE=3270

          compares the device type to the character string "3270".

          Character strings are compared independent of case.   The
          character "?" may be used in a value as a pattern
          character to match an arbitrary string of zero or more
          characters in the variable.   The character "#" may be
          used as a pattern to match any single character.   Thus,
          for example the expression

               ID=1?

          will be true for any userID beginning with "1".

               NAME = DS##

          will be true at any terminal whose name matches the first
          two characters, e.g., DS20, DS35, etc.

               NAME = ds##

          would match the same terminal names, since the case is
          ignored.

          "expression"s are composed by combining the above with
          operators &, |, and ~, with the usual meanings (and, or,
          and not).   For flexibility, a comma is treated as
          synonymous with |; thus

               VERSA,IBMPC,OP/VIS

          is equivalent to

               VERSA|IBMPC|OP/VIS

          Also, either NOT or - may be used instead of ~.

          In the absence of parentheses, the operation ~ is applied
          first, then &, then |; thus

390.6 *SIGSETUP

```
        ˜IBMPC & ID=WXYZ | TIME<500
```

is interpreted as

```
    ((˜IBMPC) & ID=WXYZ) | TIME<500
```

Parentheses may be used freely to modify or clarify order of operations.

Blanks may appear anywhere except within a keyword.

Commands

The command list consists of zero or more commands separated by /'s. If a / is required in a command, it is represented by //.

The following commands are recognized. Underlining indicates allowed minimum abbreviations.

%devicecommand The remainder of the command is a DSR
               command to be passed to the device-support
               routines via CONTROL. This may be used to
               set device options.

$mtscommand   This indicates an MTS command to be pro-
               cessed by MTS via the CMD subroutine.   Of
               course, commands which cause loading of
               another program ($RUN, $LOAD, $DEBUG,
               etc.) should not be used.

PRINT text   The remainder of the command is a message
               to be printed on SPRINT.

COMMENT text   This is the same as PRINT.

RUN file [PAR=parameter]
               The command specifies a file from which a
               program is to be dynamically loaded and
               linked to. If PAR=parameter follows the
               file name, the text following PAR is
               passed as a parameter when the program is
               called.

LINK file [PAR=parameter]
               LINK is a synonym for RUN.

CALL entryname [PAR=parameter]
               This command specifies an entry point name
               that is looked up in the loader tables and
               called if found. It may be followed by a
               parameter as for the RUN command.
```

                              This is a  rather  questionable  facility.
                              It  can  be  used  to  speed up processing
                              considerably by  link-editing  together  a
                              copy  of  SIGSETUP  with programs it is to
                              call.  This avoids opening a lot of files,
                              which  seems  to  be  what  makes  sigfile
                              processing  slow.  However, getting copies
                              of the routines is a bit dangerous.  Watch
                              out for system changes, etc.

            CUINFO guinfoparm value
                              This command may be used to set the  value
                              of any GUINFO/CUINFO parameter that can be
                              changed  by  the  user.   "guinfoparm" may
                              specify either a GUINFO item name or  item
                              number.  "value" specifies the value to be
                              assigned,  and  may  be  any  value as de-
                              scribed above.

            STOP              This  terminates  processing  of  SIGSETUP
                              input.   SIGSETUP will flush remaining in-
                              put lines until an end-of-file is  encoun-
                              tered.  It is not necessary to have a STOP
                              command; execution will terminate normally
                              on an end-of-file from SCARDS.

            entryname [PAR=parameter]
                              If  "entryname"  is  defined in the loader
                              symbol tables, a command of this  form  is
                              equivalent to

                                  CALL entryname [PAR=parameter]

                              and  causes  the  specified  routine to be
                              called.

            SIGS [id-list]
                              The SIGS command determines if the  speci-
                              fied  userIDs are currently signed on, and
                              prints out a message if so.  If  "id-list"
                              is  omitted, SIGS checks for other signons
                              of the user running SIGSETUP.  "id-list"
                              is  an  arbitrary  list of zero  or more
                              userIDs, separated by blanks  or  commas.
                              If  the  userID  running SIGSETUP  is  in
                              "id-list", SIGS checks for  other  signons
                              of that userID.

            FUNDS [parameter]
                              The  FUNDS commands prints out the remain-
                              ing  computing funds for  the  current
                              userID,  as  of  the  time of signon.  An
                              integer may be given as a  parameter.   In

this case, the output will be produced only if the remaining balance is less than the indicated amount.

### Substitutions in Commands

The value of any "variable", as described above, may be substituted into any command before it is performed. This substitution is indicated by the variable name preceded and followed by (|).

For example, |ID| is replaced by the current userID; |DATE| is replaced by the current date; |GUINFO( guinfoparm)| is replaced by the value of the specified GUINFO item. See examples below illustrating the use of substitutions in commands.

In all cases, the substituted value is converted to a suitable character form if necessary. (Note that TIME will be converted to minutes.)

Examples:     TERMINAL PRINT File space used is |GUINFO(CURRDISK)|

              DAILY & ID=WXYZ FUNDS 500

              WEDNESDAY&TOD>13:00:00 PRINT Information desk at 3:00

              TERMTYPE="IBMPC" %PAGE/%PF5=%INSERT R WXYZ:BATCH SCARDS=

              TERMINAL & TIME<1000 RUN *FORUM PAR=R

              DS## %PF3=SYS T U |id|/%FOOT=ID=|id|

              GUINFO(UNATMODE)=1 PRINT ** System in unattended mode **

              ALL PRINT Sigfile processing took |TIME| seconds.

              ALL PRINT |type| |termtype|

              ALL $SET NAME='|READ("Enter name:")|'

390.10  *SIGSETUP

| <u>*SITES</u>

Contents:      A list of the hours and the types of terminals and
               software available at each of the Campus Computing Sites.

Use:           The $COPY command should be used, e.g.,

                   $COPY *SITES

               Note:  This file replaces *WORKSTATIONS and *CCHOURS.

390.12  *SITES

January 1987

### *SLIDEQ

Contents:       The queueing program for requesting postprocessing of
                user slide descriptions.

Purpose:        To place the user's slide request(s) on the queue for
                processing.

Use:            This program is invoked by the $RUN command.

Program Key:    *SLIDEQ

Logical I/O Units Referenced:
                SCARDS - queueing requests and user input.
                SPRINT - slide receipt numbers, prompting messages for
                         queueing requests, and error messages.

Parameters:     A single queueing request may be specified in the PAR
                field of the $RUN command, if no requests are to be read
                from SCARDS.

Return Codes:   0  - Successful return.
                >0 - Unsuccessful return (program will print an error
                     message and the last queue request will not be
                     made).

Description:    Each queueing request specifies a file or tape containing
                one or more slide descriptions written in a language
                called SCODL.  *SLIDEQ reads the slide descriptions,
                places the request in the system slide queue, and charges
                the user for the slides.  At each scheduled processing
                time, the system postprocessor removes the request from
                the queue, reads the user's file (or tape), and produces
                commands for the camera.

                *SLIDEQ prompts for queueing requests using the message
                "Enter SCODL file queue request:".  The response may be a
                queueing request for a file or tape, or a line beginning
                with a "$".  If the queueing request is for a file, the
                request should consist of the file name.  Explicit
                concatenations are permitted, as are line-number ranges.
                Only permanent files may be queued; temporary files may
                not be queued.

                A response that begins with a dollar sign is treated as
                an MTS command that is to be executed immediately.  A
                null line or end-of-file response terminates execution of
                *SLIDEQ.

If the request is for a magnetic tape, the tape should already be mounted. The request should contain the pseudodevice name followed by the tape ID. Parameters must be separated by one or more blanks and/or commas. There are two optional parameters. One is the FILES parameter, which may appear anywhere after the pseudo-device name. This takes the form FILES=n, where "n" is the number of files to be plotted. The default is FILES=1. The other is the POSN parameter which may appear anywhere after the tape ID. This takes the form, POSN=string, where "string" specifies a file on the tape (any string legal for the POSN control command). If this parameter is not specified, *SLIDEQ begins reading at the current tape position (even if that is in the middle of a file). (At postprocessing, the tape will be mounted and positioned appropriately, even to the middle of a file.) *SLIDEQ and the postprocessor use whatever blocking is in effect at the time the queueing request is entered. *SLIDEQ does not rewind the tape after reading it. Pool tapes may not be queued.

If *SLIDEQ encounters an illegal SCODL command in the file being queued, it will print out an error message showing the illegal command, and the file will not be queued.

A line beginning $CONTINUE WITH will be treated as an invalid SCODL command. Implicit concatenation may not be used in plot description files (or tapes).

Queueing

After *SLIDEQ finishes reading the slide description, it prints the number of slides and the cost. The user is then asked whether the slide(s) should actually be queued. If the reply is NO, the current queue request is aborted and the user prompted for a new request (unless the request was taken from the PAR field). If the reply is YES, the slide request then is placed in the queue, a charge is made for the slide(s), and a receipt number is printed (this receipt number must be used to obtain the slide(s) at Michigan Media later). In addition, if the slide description was contained in one or more disk files, *SLIDEQ will, if necessary, permit the queued files so that the postprocessor, (a program with PKEY=*SLIDEQ) may access the plot descriptions. *SLIDEQ will establish READ access for the file to the postprocessor if necessary. (Specifically, the file will be permitted "READ PKEY=*SLIDEQ"; see the description of the $PERMIT command in MTS Volume 1, Michigan Terminal System.) A message is printed for each file so permitted. If *SLIDEQ cannot permit a file, a message is printed, and the user ensure that the file is permitted before the

slide request is postprocessed.  See the section below on postprocessing.

Postprocessing:

After a slide request has been queued, the  user's  slide description  must  be available for reading by the system postprocessor  at  the  scheduled  time.   If  the  slide description is in a file, the file must be permitted read to PKEY=*SLIDEQ.  If the file is not permitted, the slide will  be canceled.  If the plot description is on a tape, the tape  must  be  available  for  mounting  (e.g.,  not already mounted).  If the tape ID is incorrect, the slide will be canceled.

Examples:

In  the  following  examples,  terminal output appears in uppercase and user input appears in lowercase.

In the first example, the slides contained in two  files, SLIDE  (containing  one slide) and MORESLIDES (containing two slides), are to be queued.

```
#$run *slideq
#run *slideq
#Execution begins

 Enter SCODL file queue request:
 slide

 File "WXYZ:SLIDE" has been permitted
 "READ PKEY=*SLIDEQ".

 One slide queued for file "WXYZ:SLIDE".
 Cost will be: $4.50
 Okay?
 ok
 SCODL file assigned reciept number: 780121

 Enter SCODL file queue request:
 moreslides

 File "WXYZ:MORESLIDES" has been permitted
 "READ PKEY=*SLIDEQ".

 2 slides queued for file "WXYZ:MORESLIDES".
 Cost will be: $9.00
 Okay?
 ok
 SCODL file assigned reciept number: 780122

 Enter SCODL file queue request:
```

```
        Total of 3 slides queued this run.

    #Execution terminated
```

In the second example, the user does not have appropriate access to WABC:SLIDES for *SLIDEQ to permit the file. Note that the user must make sure that the file is permitted before the next postprocessing time.

```
    #run *slideq par=wabc:slides
    #Execution begins

     Unable to permit file "WABC:SLIDES".
     Permit file "WABC:SLIDES" to "READ PKEY=*SLIDEQ"
      before next queue processing time, or the file
      will not be able to be processed.

     One slide queued for file "WABC:SLIDES".
     Cost will be: $4.50
     Okay?
     ok
     SCODL file assigned reciept number: 780123

     Total of one slide queued this run.
    #Execution terminated
```

For more information on producing slides, see Computing Center Memo 475, "Producing Slides on MTS."

January 1987

*SNOBOL4, *SNOBOL4B

Contents: The SNOBOL4 interpreter (version 3).

Purpose: To interpret SNOBOL4 source programs.

Use: The program is invoked by the $RUN command.

Program Key: *SNOBOL4
               *SNOBOL4B

Logical I/O Units Referenced:
               5   - source for the SNOBOL4 program to be translated,
                          immediately followed by data to be read by the
                          default INPUT string.
               6   - output from compilation of the SNOBOL4 program
                          and output via the default OUTPUT string.
               7   - output via the default PUNCH string.

Parameters: The following parameters may be specified in the PAR
               field of the $RUN command.   The parameters must be
               separated by a comma.

               DUMP   - causes a SNOBOL4 string dump only if a fatal
                            error occurs during execution.
               XREF   - causes a cross-reference dictionary of the
                            source   program   to   be   printed after
                            compilation.
               NOEX   - prevents execution of a SNOBOL4 program if
                            any compilation errors are present.
               SIZE=xxxx - specifies the amount of storage to be used
                            for the compiled SNOBOL4 program and its
                            variables (the SNOBOL4 interpreter itself
                            uses about 40 pages). "xxxx" may be any
                            number specifying the number of pages to be
                            reserved or the word MIN may be used to
                            specify that 10 pages be reserved. If no
                            SIZE parameter is present, the default value
                            is 30 pages.

Return Codes: Always zero.

Description: The language accepted by *SNOBOL4 is described in the
               publication, The SNOBOL4 Programming Language, by
               R. Griswold, J. Poage, and I. Polonsky (Prentice-Hall,
               1971).

               The file *SNOBOL4B contains a version of the SNOBOL4
               interpreter that supports the BLOCKS feature.

The file *SPITBOL contains a faster and more economical version of SNOBOL4. This is recommended for general SNOBOL4 use.

For further details on the use of the SNOBOL language in MTS, see MTS Volume 9, SNOBOL4 in MTS.

Examples: $RUN *SNOBOL4 5=*SOURCE* 6=*SINK*

                 In the above example, the source program and data are read from *SOURCE* and the output is written to *SINK*.

                 $RUN *SNOBOL4 5=SNOPROG+*SOURCE* 6=*DUMMY*(1,20)+*SINK* 7=*PUNCH*

                 In the above example, the source program is read from the file SNOPROG and the data is read from *SOURCE*. The first twenty lines of output are discarded with the remainder being written to *SINK*. Punch output is written to *PUNCH*.

January 1987

<u>*SNOSTORM</u>

Contents: The SNOSTORM preprocessor program.

Purpose: To preprocess SNOSTORM source programs into SPITBOL
                 source programs and then to invoke the SPITBOL compiler
                 to produce an object module.

Use: SNOSTORM is invoked by the $RUN command.

Program Key: *SNOSTORM

Logical I/O Units Referenced:
                 SCARDS - SNOSTORM source program input.
                 SPRINT - Listing output, including diagnostics.
                 SPUNCH - Object deck output. The object includes a
                          $CONTINUE WITH *SPITLIB record at the end.
                 SERCOM - Diagnostics.

                 Temporary files created:

                     Two temporary files are created in the process of
                     translating a SNOSTORM program. Both of these files
                     are emptied before they are used.

                     -PRINT is a file which will contain the SPITBOL
                     listing of the program produced by SNOSTORM. This
                     file name is chosen to correspond to the file that
                     *SPITDEBUG uses in printing source statements when
                     errors are encountered.

                     -SNOOBJ is the file that is used to hold the SPITBOL
                     program produced by SNOSTORM. This is an intermedi-
                     ate file that is then used as input to the SPITBOL
                     compiler.

Parameters: Options may be specified in the PAR field of the $RUN
                 command. If more than one option is specified, they
                 should be separated by commas with no intervening blanks.

                     SIZE=n   The SIZE parameter sets the amount of
                                     memory used by SNOSTORM and SPITBOL in the
                                     process of compiling the source program.
                                     It may be necessary to increase this
                                     parameter for large source programs. The
                                     value of <u>n</u> may not be less than 20 nor
                                     more than 256. If specified, this parame-
                                     ter must be placed before other parame-
                                     ters. The default is SIZE=20.

COM/NOCOM   The COM option controls the passage of
                   comments into the target module.   If COM
                   is specified, all comments in the source
                   module are passed into the target module.
                   If NOCOM is specified, no comments are
                   passed to the target module. The default
                   is NOCOM.

                   CASE statements will also generate com-
                   ments in the target module if the COM
                   option is in effect.

INDENT='string'/NOINDENT
                   The INDENT option is followed by an '='
                   and a quoted string. This string is taken
                   as the string to be replicated once for
                   each indentation level. Automatic inden-
                   tation may be turned off with the NOINDENT
                   option. The default is INDENT='. '.

LIST/NOLIST   The LIST option controls all   further
                   source program listing, either suppress
                   ing it with NOLIST or enabling it with
                   LIST.   It performs the same action as the
                   LIST {ON|OFF} statement. The default is
                   LIST.

CONVERT   The CONVERT option converts obsolete SNOS-
                   TORM statements   into a currently accept-
                   able form. When this option is specified,
                   SNOSTORM will read the old program from
                   SCARDS and write a new version of the
                   program on SPUNCH, preserving the line
                   numbers above 1.000.   SNOSTORM will not
                   translate during this process except for
                   translating obsolete forms to new forms.

DEBUG/NODEBUG   The DEBUG option is provided to aid in
                   using *SPITDEBUG. The DEBUG option causes
                   extra code to be inserted in the SPITBOL
                   program to allow *SPITDEBUG breakpoints to
                   be set at any statement. Execution of the
                   program will begin in *SPITDEBUG so that
                   there is an opportunity to issue debugging
                   commands before the program is run.

                   The extra code generated by the DEBUG
                   option will require extra CPU time and
                   memory space in the   object   program.
                   Because the number of statements (but not
                   the CPU time) is approximately tripled, it
                   may be necessary to increase the value of
                   &STLIMIT.

Without the DEBUG option, *SPITDEBUG is still included in the program, but no extra code is generated for breakpoints or to begin execution in the debugger. To suppress the inclusion of *SPITDEBUG, the NODEBUG option must be specified.

Return Codes: Always zero.

Description: The SNOSTORM preprocessor was developed at the Computing Center to facilitate structured programming in SPITBOL, a dialect of SNOBOL4. For a full description of SNOSTORM, see MTS Volume 9, <u>SNOBOL4 in MTS</u>.

400 *SNOSTORM

January 1987

*SOFTWARECHARGES

Contents: A list of the programs for which program-product sur-
              charges are made.

Use: The list may be obtained via the $COPY command, e.g.,

                  $COPY *SOFTWARECHARGES

January 1987

*SORT

Contents: The sort/merge/blocking program.

Use: The program is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:
                 GUSER - a control statement, which is terminated by the
                          END parameter or an end-of-file, is read via
                          GUSER if one is not passed as a parameter.
                 SCARDS - input records are read from SCARDS if an input
                          data set name is omitted from the control
                          statement.
                 SERCOM - prompting and sort/merge program diagnostics.
                 SPUNCH - output records are written to SPUNCH if an
                          output data set name is omitted from the control
                          statement.

Return Codes: Return code from SORTE9 (if SORTE9 provided), or

                 0 - Successful return.
                 4 - Error return.

Description: The sort/merge program

                 (1) orders records according to the collating se-
                      quence defined in its control statement;
                 (2) provides blocking/deblocking facilities, proces-
                      sing data sets of types U, F, V, VS, FB, FBS, VB,
                      and VBS;
                 (3) allows user subroutines to gain control at sever-
                      al points, permitting the generation, modifica-
                      tion, deletion, and comparison of records.

                 The following description is designed to familiarize
                 users with certain aspects of the control statement and
                 to enable them to run simple sorts using the most common
                 type of ordering: character, or alphabetized, collation.
                 It is not intended to describe all, or even the best,
                 ways to do simple character sorts.   Rather, it will
                 describe one way which produces the desired results.

                 The sort program is invoked by entering a $RUN *SORT
                 command. The control statement, which describes the
                 processing of the data, is placed in the PAR field of the
                 $RUN command.   This control statement defines how the
                 output is to be ordered and the locations of the input
                 and output data.   The following paragraphs describe a

control statement for a simple character sort on quanti-
ties of data up to at least the lesser of 225,000 records
and 18,000,000 characters.

The first parameter to be specified in the PAR field is
the keyword "SORT=". It is then necessary to specify the
criteria for ordering the data. This is done by follow-
ing "SORT=" with the positional parameters "CH,A,". The
"CH" means that the ordering is to be based on character
data, that is, data composed of alphanumeric characters
or numbers in character form. Character data are pro-
duced, for example, by keypunches, terminals, and for-
matted output from FORTRAN. The "A" means that the
output is to be in ascending order, e.g., the characters
in the string " abzAZ019" are ascending. The $RUN
command should now look like this:

        $RUN *SORT PAR=SORT=CH,A,

Note that a blank must not immediately precede any equal
sign or comma in the control statement.

The location of the characters which are to determine the
ordering of the records must now be supplied. Suppose
that the field on which the sort is to be based starts in
position, or column, 15 and ends in position 20 in each
record. This is indicated to SORT as shown in the
example below. The "A," in the control statement is
followed by the starting position, "15" in this case, and
the number of characters, or length, of the field, "6" in
this case. The two values are separated by a comma. The
$RUN command is now

        $RUN *SORT PAR=SORT=CH,A,15,6

If the designated fields (positions 15-20 in the example)
of two records are compared and found to be identical, it
may be desired to compare a second field in each record
to determine which record should precede the other. This
is accomplished by appending a comma and another CH,A,
starting-position,number-of-characters combination to the
first description. Suppose that the second field occu-
pies positions 79-80 of the records. The $RUN command
would then be

        $RUN *SORT PAR=SORT=CH,A,15,6,CH,A,79,2

Additional fields for the resolution of identical com-
parisons may be added in the same manner.

The next step is to specify where the input is to be
found. This is done with the keyword "INPUT=" followed

by a file or pseudodevice name, e.g., IN(5,99)@-TRIM or
I. The $RUN command might now be

```
$RUN *SORT PAR=SORT=CH,A,15,6,CH,A,79,2
    INPUT=IN(5,99)@-TRIM
```

(In the case of blocked magnetic tapes, this is neither
the most efficient nor the recommended method of com-
pletely specifying the input data set to SORT; but if
tape blocking is left on, which is the default, it does
work. It should also be noted that SORT does not
reposition tapes either before or after using them nor
does it write end-of-file marks.)

Finally, the place where SORT is to write the sorted
output must be designated. The keyword "OUTPUT=" fol-
lowed by a file or pseudodevice name accomplishes this.
If the output is to be written to a tape with the
pseudodevice name *O*, the $RUN command would be

```
$RUN *SORT PAR=SORT=CH,A,15,6,CH,A,79,2
    INPUT=IN(5,99)@-TRIM OUTPUT=*O*
```

(The comments regarding magnetic input tapes also apply
to tapes used for output). The use of the same data set
for both input and output should not even be considered
unless the data would be easily recoverable if it were
damaged and unless the user is knowledgeable about both
SORT and the characteristics of the data set.

An alternate way of specifying the input and output data
sets is to assign the input data set to SCARDS and the
output data set to SPUNCH. Using this alternative, the
$RUN command for the same sort as above would look like:

```
$RUN *SORT SCARDS=IN(5,99)@TRIM SPUNCH=*O*
    PAR=S=CH,A,15,6,CH,A,79,2
```

When SORT finishes, it prints two statistics separated by
a slash. The first of these is the number of records
read by SORT. The second, which may normally be ignored,
is the number of scratch files used by SORT.

For a complete description of the sort/merge program, see
"The SORT Utility Program" description in MTS Volume 5,
System Services.

Examples: The following include features not described above.

```
$RUN *SORT PAR=C I=*TAPE*,VB,50,32767 O=-T
```

              In the above example, the contents of *TAPE* are
              copied into the file -T. *T* has VB-formatted

records with a maximum record length of 50 bytes and a maximum block length of 32,767 bytes. The records written to -T will have U format and maximum record and block lengths of 50 bytes. Assuming *TAPE* is a magnetic tape, blocking by the tape routines must be disabled.

```
$RUN *SORT SCARDS=*T* SPUNCH=*SINK* PAR=S=C,D,5,15 R=4000
```

In the above example, the contents of *T*, which is approximately 4000 records, will be written onto the current sink data set, ordered such that positions 5-19 of each record form a descending character sequence. Both *T* and the sink are treated as having U-formatted records. Assuming *T* is a magnetic tape, the maximum input record length will be the value of the SIZE parameter (if blocking is off) or the LRECL parameter (if blocking is on); the maximum output record length will be equal to the maximum input record length.

January 1987

Summary of the Control Statement

Prototype:

```
[COPY|[[SORT|MERGE][=[[type],[aspect],[location],[length],]...
             [type][,[aspect][,[location][,[length]]]]]]]]
[[DS=delimiter[string]delimiter▪]...DS=delimiter[string]delimiter]
[INPUT[=[[name],[structure],[record length],[block length],]...
             [name][,[structure][,[record length][,[block length]]]]]]]]
[OUTPUT[=[[name],[structure],[record length],[block length],]...
             [name][,[structure][,[record length][,[block length]]]]]]]]
[additional parameter]...  [END ]
```

Collating fields:

TYPE	CODE	SIGN PRESENT	FIELD LENGTH (BYTES)
alignment	AL	no	1 - 4095
binary	BI	no	1 - 256
bit	BT	no	1 - 255 (mask)
call	CA	-	-
character	CH	no	1 - 256
defined sequence	DS(i)	no	1 - 256
fixed-point	FI	yes	1 - 260
floating-point	FL	yes	2 - 16
length	LE	-	-
packed decimal	PD	yes	1 - 16
sequence	SE	-	-
signed decimal	SD	yes	1 - 17
zoned decimal	ZD	yes	1 - 16

Record structures:

CODE	RECORD STRUCTURE
U	undefined length
F	fixed length
V	variable length
VS	variable length; spanned
FB	fixed length; blocked
VB	variable length; blocked
VBS	variable length; blocked; spanned
FBS	fixed length; blocked; standard

Additional parameters:

```
    CHK    (exit check facility)
    DEC    (delete comments)
    DEL=x[,x]...  (delete output records)
    HELP [topic]  (enter help facility)
    LIO    (list data set characteristics)
    {REC|MNR}=x  (number of records)
    RES=x  (restart)
    SIG    (sign off on error)
    TPS[={x|name,name[,name]...}]   (tape-merge sort facility)
```

January 1987

<u>*SPIRES</u>

Contents: The SPIRES database management system.

Purpose: To define, search, and modify data bases.

Use: This program is invoked by the $RUN command.

Program Key: *SPIRES

Logical I/O Units Referenced:
                SCARDS - SPIRES commands.
                SPRINT - output and error messages.

Description: SPIRES (<u>S</u>tanford <u>P</u>ublic <u>I</u>nformation <u>R</u>etrieval <u>S</u>ystem) is
                available in MTS.   SPIRES is a computer information
                storage and retrieval system. It is the product of an
                intensive development effort at Stanford, and represents
                a significant advance in information   retrieval
                technology.

                SPIRES allows a user to create, update, maintain, inter-
                rogate, and display stored information. This information
                may be kept in either the user's own private database of
                records or in a public database. As an on-line interac-
                tive system, SPIRES enables the user to communicate with
                the database in a conversational mode. SPIRES is general
                enough to handle most forms of data -- numeric, textual,
                and codified. It supports a wide spectrum of applica-
                tions, ranging from management and administrative support
                to bibliographic reference retrieval and handling of
                numeric and codified information. Because of its flexi-
                bility, it can easily be adapted to changes in a user's
                requirements.

                To find out how to obtain documentation for SPIRES, the
                user may issue the command

                    $COPY DBMD:DOCUMENTATION

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command.   The parameters must be
                separated by commas and/or blanks. The minimum accept-
                able abbreviations are underlined.

                    <u>T</u>ERSE

                        The TERSE parameter will cause a shorter greeting
                        message to be printed when the program   is
                        invoked.

ACTIVE=filename

        The ACTIVE parameter will set the file "filename" as the active file. "filename" must be a line file. If omitted, the file -SPIRES will be used as the active file.

{SPIBILD|SPICOMP|FASTBILD|SPIRES}

        These parameters specify the processor to be invoked. If omitted, the SPIRES processor is invoked.

Return Codes: None.

January 1987

<u>*SPITBOL</u>

Contents: Version 2 of the SNOBOL4 compiler developed at the
                 Illinois Institute of Technology.

Purpose: To compile and execute SNOBOL4 source programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *SPITBOL

Logical I/O Units Referenced:
                 SCARDS - SNOBOL4 source program to be compiled followed
                          by the data read via the default variable INPUT.
                 SPRINT - source listing, object listing, parameter list-
                          ing, compiler diagnostics, execution-time diag-
                          nostics, and output via the default variable
                          OUTPUT.
                 SPUNCH - object module if the DECK option was specified,
                          and output from the default variable PUNCH.
                 0   - object module if NODECK and LOAD parameters are
                          specified.
                 SERCOM - prompting if a terminal.
                 GUSER - user responses to prompting if a terminal.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command.   The parameters must be
                 separated by a comma or by one or more blanks.   A
                 semicolon may be used to terminate the PAR field; the
                 text after the semicolon is not processed by the SPITBOL
                 compiler but is available to the program via the SYSPAR
                 external function. The parameters may be either keywords
                 or free verbs. Free verbs may be negated by one of three
                 prefixes: "NO", "˜", or "-". The underlined portion of
                 the parameter may be used as the minimum acceptable
                 abbreviation. The legal parameters are:

                 <u>ALIST</u> or <u>NOA</u>LIST

                     If ALIST is specified, an object-code listing will
                     be produced on SPRINT.   The default is NOALIST.
                     This parameter replaces the OLIST parameter.

                 <u>BATCH</u> or <u>NOBA</u>TCH

                     If BATCH is specified, the compiler will batch
                     process input decks. The batch pseudo-end-of-file
                     is "./*" in columns 1-3. The default is NOBATCH.

CSTAT or NOCSTAT

    If CSTAT is specified, compilation statistics are printed on SPRINT. The default is NOCSTAT.

DECK or NODECK

    If DECK is specified, an object module will be produced on SPUNCH. SPITBOL object modules must be run in concatenation with *SPITLIB. The default is NODECK.

DUMP=nnn

    At termination of execution, the SPITBOL dump function is called with "nnn" as the argument. The default is DUMP=0 which produces no dump.

EDUMP=nnn

    If execution terminates abnormally, the SPITBOL dump function will be called with "nnn" as the argument. If SPRINT is assigned to a terminal, the default is EDUMP=0 which produces no dump; if SPRINT is not assigned to a terminal, the default is EDUMP=1 which generates a dump of natural variables and keywords.

ERRXEQ or NOERRXEQ

    If ERRXEQ is specified, the compiled program is executed even if errors were detected during compilation. The default is ERRXEQ.

ESTAT or NOESTAT

    If ESTAT is specified, execution statistics are produced on SPRINT. The default is NOESTAT.

EXECUTE or NOEXECUTE

    If EXECUTE is specified, the compiled program is executed. The default is EXECUTE.

FAILCHK or NOFAILCHK

    If FAILCHK is specified, execution is terminated with an error if a statement is executed that fails and there is no conditional "goto" field. This is a useful debugging tool. The default is NOFAILCHK.

January 1987

INMGN=nnn

"nnn" is the right-hand margin for the compiler when scanning input source programs. The default is set to the minimum of the input record length for SCARDS and 255. Programs that have sequential IDs in columns 73-80 should compile with INMGN=72.

LINECNT=nnn

"nnn" is the number of lines per page for printed output. This must be in the range of 3 ≤ nnn ≤ 32767. The default is 58.

LIST or NOLIST

If LIST is specified, a source listing of the compiled program is printed on SPRINT. The default is LIST unless SPRINT is assigned to a terminal. This parameter replaces the SLIST parameter.

LOAD or NOLOAD

If both LOAD and NODECK are specified, an object module will be produced on logical I/O unit 0. The default is NOLOAD.

OPTIMIZE or NOOPTIMIZE

If OPTIMIZE is specified, the object code produced will be optimized. This can reduce the CPU time and storage required for program execution. The default is OPTIMIZE.

PLIST or NOPLIST

If PLIST is specified, a list of parameter values will be printed on SPRINT. The default is NOPLIST.

SDUMP or NOSDUMP

If SDUMP is specified, a storage dump of the SPITBOL work areas will be produced on SPRINT if an internal SPITBOL error is detected. This is useful only for system debugging. The default is NOSDUMP.

SEQCHK or NOSEQCHK

If SEQCHK is specified and INMGN=72, the sequential ID field (columns 73-80) is checked for ascending order. Out-of-order cards will be flagged. The default is NOSEQCHK.

SIZE=nnn

"nnn" is the number of pages allocated for dynamic
storage within the compiler. This must be in the
range of 4 ≤ nnn ≤ 256. The default is 20.

TIMECHK or NOTIMECHK

If TIMECHK is specified, SPITBOL will terminate
execution of the user program 0.25 seconds before
any specified time limit in order that a symbolic
dump may be given. The default is TIMECHK.

XREF or NOXREF

A cross-reference listing of the symbols in the
compiled program is printed on SPRINT. The default
is NOXREF.

Return Codes: Always zero.

Description: *SPITBOL is a SNOBOL4 compiler which is considerably
               faster than *SNOBOL4, and can generate object modules if
               desired.   However, the object modules must be run with
               *SPITLIB. For further details, see the section "SPITBOL"
               in MTS Volume 9, SNOBOL4 in MTS.

               The file *SPITERR contains the SPITBOL error messages.
               Each message is located at an MTS line number correspond-
               ing to the SPITBOL error number for that message. The
               message may be obtained via the $COPY command, e.g.,

                   $COPY *SPITERR(11.009,11.009)

               prints SPITBOL error message 11.009.

Examples: $RUN *SPITBOL SCARDS=PROG.S PAR=SIZE=50

               In the above example, the program in the file PROG.S
               is compiled and executed with a dynamic storage size
               of 50 pages.

               $RUN *SPITBOL SCARDS=PROG.S SPRINT=*PRINT* SPUNCH=PROG.O
                   PAR=DECK,NOEX

               In the above example, the program in PROG.S is
               compiled without being executed. The object module
               is written into the file PROG.O; the source listing
               is produced on *PRINT*.

January 1987

*SPITDEBUG

Purpose: To assist in the interactive debugging of SPITBOL programs.

Use: The following statement should be included in the SPITBOL source program at a place where it will be executed at the beginning of the program starting at column one:

              -COPY *SPITDEBUG

              If a copy of the SPITBOL-generated source listing is placed in the file -PRINT*, *SPITDEBUG will print the statement causing the error.

Program Key: *EXEC

Description: *SPITDEBUG contains a number of SPITBOL source statements that are invoked whenever an execution error occurs or the DEBUG function is called.

              Errors are trapped using the SETEXIT function and setting &ERRLIMIT to 1. When SPITDEBUG gains control after an error, the error number and associated error comment are printed. If the statement can be found in the file -PRINT*, it is also printed. After these messages are printed, the user is placed in SPITDEBUG command mode where he may enter SPITDEBUG commands or SPITBOL statements.

              If the DEBUG function is called, it may have a string as an argument which is printed at the time of the call. These calls may be placed in the source program in order to set a breakpoint and examine the status of variables using SPITDEBUG. Processing may be continued with the SPITDEBUG CONTINUE command.

              The SPITDEBUG commands are given below. The underlined portion of each command is the minimum acceptable abbreviation that may be used.

                  BREAK label   Sets an execution breakpoint at "label". This breakpoint will be effective only on a goto to that label; it is not effective if execution of the previous statement falls into that label. Labels may be generated automatically on all statements via the SNOSTORM DEBUG option. The breakpoint feature is

implemented with the SPITBOL TRACE function and the keyword &TRACE is incremented by 100 each time a BREAK command is issued. This inadvertently may enable other unrelated tracing in the program. When used with programs compiled with the SNO-STORM DEBUG option, it is possible to use the source-file line number in place of the label.

CONTINUE   Resumes program execution.

DISPLAY express Displays the program value "express", e.g.,

         DISPLAY x<2>
         x<2> = "HELLO"

DUMP {1|2}   Produces a SNOBOL dump (DUMP(1) or DUMP(2)).

EXPLAIN   Synonym for HELP.

GOTO label   Resumes program at the location "label"; this is the same as ":(label)" in the program.

         GOTO START

HELP   Provides a list of the legal SPITDE-BUG commands with a brief description of each.

IGNORE ["msg"] n Causes the next "n" calls on the DEBUG function with parameter "msg" to be ignored. If "msg" is omitted, DEBUG calls with a null string parameter will be ignored.

MTS [command] Enters MTS command and optionally executes an MTS command, e.g.,

         MTS $EDIT MYFILE

REMOVE label   Removes the breakpoint at "label".

RESTORE label Synonym for REMOVE.

STOP   Terminates execution.

$command   Executes an MTS command.

January 1987

*SPITLIB

Contents: The version 2 execution-time support routines for pro-
                grams compiled by *SPITBOL.

Use: *SPITLIB should be concatenated with the object file to
                be executed on the $RUN command, i.e.,

                    $RUN object+*SPITLIB

Program Key: *EXEC

Logical I/O Units Referenced:
                SCARDS - data to be read via the default variable INPUT.
                SPRINT - execution-time diagnostics, and output via the
                         default variable OUTPUT.
                SPUNCH - output via the default variable PUNCH.
                SERCOM - prompting if a terminal.
                GUSER - user responses to prompting if a terminal.

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command. The parameters must be
                separated by a comma or by one or more blanks. A
                semicolon may be used to terminate the PAR field; the
                text after the semicolon is not processed by the SPITBOL
                compiler but is available to the program via the SYSPAR
                external function.   The parameters may be keywords or
                free verbs. Free verbs may be negated by one of three
                prefixes: "NO", "˜", or "-". The underlined portion of
                each parameter may be used as the minimum acceptable
                abbreviation. The legal parameters are:

                DUMP=nnn

                    At termination of execution, the SPITBOL dump func-
                    tion is called with "nnn" as the argument.   The
                    default is DUMP=0 which produces no dump.

                EDUMP=nnn

                    If execution terminates abnormally, the SPITBOL dump
                    function will be called with "nnn" as the argument.
                    If SPRINT is assigned to a terminal, the default is
                    EDUMP=0 which produces no dump; if SPRINT is not
                    assigned to a terminal, the default is EDUMP=1 which
                    generates a dump of natural variables and keywords.

              ESTAT or NOESTAT

                  If ESTAT is specified, execution statistics are
                  generated on SPRINT. The default is NOESTAT.

              LINECNT=nnn

                  "nnn" is the number of lines per page for printed
                  output. This must be in the range of 3 ≤ nnn ≤
                  32767. The default is 58.

              PLIST or NOPLIST

                  If PLIST is specified, a list of parameter values is
                  printed on SPRINT. The default is PLIST.

              SDUMP or NOSDUMP

                  If SDUMP is specified, a storage dump of the SPITBOL
                  work areas will be produced on SPRINT if an internal
                  SPITBOL error is detected. This is useful only for
                  system debugging. The default is NOSDUMP.

              SIZE=nnn

                  "nnn" is the number of pages of dynamic storage
                  allocated. This must be in the range of 4 ≤ nnn ≤
                  256. The default is 20.

              TIMECHK or NOTIMECHK

                  If TIMECHK is specified, SPITBOL will terminate
                  execution of the user program 0.25 seconds before
                  any specified time limit in order that a symbolic
                  dump may be given. The default is TIMECHK.

Description: Object modules generated by *SPITBOL must be run with the
              execution-time support routines in *SPITLIB. For further
              details, see the section "SPITBOL" in MTS Volume 9,
              SNOBOL4 in MTS.

Examples: $RUN PROG.OBJ+*SPITLIB SCARDS=DATA1

                  In the above example, the program in the file
                  PROG.OBJ is executed with the data being read from
                  the file DATA1.

              $RUN -OBJ+*SPITLIB 5=INFILE 6=OUTFILE PAR=SIZE=50

                  In the above example, the program in the file -OBJ
                  is executed with input from the file INFILE and
                  output being written to the file OUTFILE.

January 1987

<u>*SYMBOLS</u>

Contents: A program to print a listing of all external symbols
                within the MTS system.

Use: The program is invoked by the $RUN command.

Program Key: *SYMBOLS

Logical I/O Units Referenced:
                SCARDS - the specified file or device from which the
                         control record is read.
                SPRINT - the specified file or device on which the output
                         listing is produced.

Return Codes: Always zero.

Description: This program produces the listing of MTS external symbols
                in the system. The listing is produced by searching the
                external symbol dictionaries of the system and the
                library.   Hence, it is current as of the instant it is
                run.

January 1987

*SYSMAC

Contents: The standard macro library for use with the MTS 360/370
                assemblers *ASMG and *ASMH.

Purpose: To supply standard macro definitions for frequently used
                MTS facilities.

Use: The file should be assigned to one of the logical units
                0, 2, 3, 4, or 5 on the $RUN *ASMG or $RUN *ASMH
                commands. For *ASMH, the default is *SYSMAC if unit 0 is
                not assigned.

Program Key: *EXEC

Description: The form of a macro library and descriptions of the
                macros in *SYSMAC are given in MTS Volume 14, 360/370
                Assemblers in MTS.

Example: $RUN *ASMH SCARDS=PROGRAM 0=*SYSMAC
                $RUN *ASMH SCARDS=PROGRAM

                In the above examples which are equivalent, the
                macro library for the program in the file PROGRAM is
                assigned to logical I/O unit 0.

January 1987

*TANGO

Contents: The object module of the Tango compiler.

Use: The Tango compiler is invoked by the $RUN command.

Program Key: *TANGO

Logical I/O Units Referenced:
                SCARDS - the Tango source program.
                SPRINT - a source program listing.
                SPUNCH - the object module if there are no compilation
                         errors.

Parameters: The following parameters may be speciified in the PAR
                field of the $RUN command.

                CHECK/NOCHECK

                    The CHECK option causes certain run-time checks to
                    be produced in the object module.   The NOCHECK
                    option suppresses these checks.   The default is
                    CHECK.

                LIST/NOLIST

                    The LIST option produces a listing of the source
                    program on SPRINT. The NOLIST option suppresses the
                    source program listing. The default is LIST.

                PRINT_CODE/NOPRINTCODE

                    The PRINT_CODE option produces a listing of the
                    object code on SPRINT.   The NOTPRINT_CODE option
                    suppresses   this   listing.   The   default   is
                    NOPRINT_CODE.

Return Codes: 0 - Successful return.
                4 - Warning messages printed.
                8 - Errors detected.

Description: The Tango compiler, developed at the University of
                Michigan Computing Center, is designed to be a modern
                replacement for Snobol4 and Spitbol. It has the pattern
                matching and table facilities of Snobol and is augmented
                by structured-programming features and strong typing
                similar to those in Pascal and Modula.

                The run-time library *TANGOLIB must be concatenated to
                the object file when it is executed.

Example: $RUN *TANGO SCARDS=PROG.SOU SPUNCH=PROG.OUT PAR=NOLIST
                  $RUN PROG.OUT+*TANGOLIB

                  In the above example, the Tango source program in
                  the file PROG.SOU is compiled and subsequently
                  executed.

January 1987

<u>*TAPECOPY</u>

Contents: The MTS tape-copying program.

Purpose: To copy magnetic tapes.

Use: The program may be invoked with the $RUN command or may
                be called as a subroutine from a user-supplied program.
                In the latter case, the user must concatenate *TAPECOPY
                with the FDname(s) containing the object modules of his
                own program, e.g.,

                    $RUN object+*TAPECOPY

Program Key: *TAPECOPY

Logical I/O Units Referenced:
                SPRINT - the listing of the number of blocks or records
                         in each file.
                SERCOM - error comments and prompting messages.
                GUSER - input and output tape pseudodevice names.
                0   - the input tape pseudodevice name.
                1   - the output tape pseudodevice name.

Parameters: The following parameters may be specified in the PAR
                field of the $RUN command.   The parameters must be
                separated by commas.

                FILES=n   The number of files (or data sets) to be
                            copied is specified by "n".
                RECORDS=m   The number of blocks or records (see REBLK
                or BLOCKS=m parameter) to be copied in file n+1 is
                            specified by "m", where "m" is a one to six
                            digit decimal integer. If "n" and "m" are
                            both zero or omitted, then all files are
                            copied (see below).
                NOREW   The tapes are not rewound before or after
                            copying.
                OMITERRORS Whenever a permanent read error is encoun-
                            tered on the input tape, an error comment
                            (indicating the file and record in error)
                            will be printed on SERCOM.   The block in
                            error will be skipped and copying will con-
                            tinue with the next block.
                COPYERRORS This parameter causes the same error comment
                            as OMITERRORS. However, any blocks which are
                            in error <u>are</u> copied.   Any block so copied
                            will no longer contain the error indication;
                            however, the validity of the data in any such
                            block is unpredictable.

REBLK	This parameter will cause blocking to be enabled for both the input and output tapes so that records, rather than blocks, are copied. Normally, blocking is disabled so that entire blocks are copied (for efficiency).
REW	The tapes are rewound before and after copying.

Return Codes: 0 - Successful return.
               4 - Invalid parameter.
               8 - Fatal error.

Description: The complete description of *TAPECOPY is given in MTS Volume 19, <u>Tapes and Floppy Disks</u>.

January 1987

<div align="center">*TAPEDUMP</div>

Contents: The MTS tape (or file) dumping program.

Purpose: To dump a magnetic tape or disk file in both character
                 and binary formats.

Use: The program is invoked by the $RUN command.

Program Key: *TAPEDUMP

Logical I/O Units Referenced:
                 SPRINT - dump output.
                 SERCOM - prompting and error messages.
                 GUSER - tape or file to be dumped if unit 0 is not
                          assigned.
                 0   - magnetic tape or file to be dumped.

Parameters   The following parameters may be specified in the PAR
                 field of the $RUN command.   The parameters must be
                 separated by commas.

                 FILES=n   Specifies the number of files to be dumped.

                 RECORDS=m   Specifies the number of records to be dumped.

                 NODUMP   Suppresses the printing of the actual dump.

                 EBCDIC   Prints the dump in hexadecimal/EBCDIC.

                 BCD   Prints the dump in octal/BCD.

                 CV   Forces the reading of 7-track tapes with the
                             data converter feature enabled.

Return Codes: Always zero.

Description: The complete description of *TAPEDUMP is given igned to
                 in MTS Volume 19, _Tapes and Floppy Disks_.

January 1987

<u>*TAPEFIXER</u>

Contents: A program to recover data from a 9-track labeled tape
                 which has been partially destroyed or contains unreadable
                 blocks.

Use: The program is invoked by the $RUN command.

Program Key: *TAPEFIXER

Logical I/O Units Referenced:
                 SPRINT - labels encountered on the input tape (see the
                          parameter descriptions below).
                 SERCOM - diagnostic and prompting messages.
                 0   - input tape to be scanned. This must be a
                          9-track tape and cannot be a pool tape since
                          label processing cannot be disabled for pool
                          tapes.
                 1   - corrected version of input tape. This must be a
                          9-track tape and may be either labeled or
                          unlabeled. The tape need not be positioned at
                          the load point. The tape must be mounted with
                          RING=IN specified on the mount request.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command.

                 LABELS   Labels encountered on the input tape will be
                           echoed on SPRINT. This is the default.

                 <u>NOL</u>ABELS Labels encountered on the input tape will not
                           be echoed on SPRINT.

                 NOTEXT   Data records are not copied to the output tape.

                 TEST   When searching for a label, all blocks are
                           inspected, not just the blocks that are exactly
                           80 bytes long.

Return Codes: 0 - Successful return.
                 8 - Error return.

Description: The complete description of *TAPEFIXER is given in MTS
                 Volume 19, <u>Tapes and Floppy Disks</u>.

430 *TAPEFIXER

*TAPES

Contents: A magnetic-tape status program.

Purpose: To allow magnetic-tape users to PERMIT, RENAME, and find
                out the status of magnetic tapes stored by the Computing
                Center.

Use: The program is invoked by the $RUN command.

Program Key: *TAPES

Return Codes: Always zero.

Description: *TAPES is a program for changing the name and permission
                status of user-owned magnetic tapes, and for displaying
                the results of those changes.

Commands: The following commands are available for *TAPES. The
                underlined portion of each command name may be used as an
                abbreviation.

                STATUS [tapename] [options ...]

                    The STATUS command gives information about a user's
                    own tapes and of tapes of other users that are
                    accessible. The syntax of this command is very
                    similar to the MTS $FILESTATUS command.   If no
                    parameter is given, information is given about all
                    tapes.   If "tapename" is specified, information is
                    given about a specific tape or set of tapes.
                    "tapename" may be the name of a tape (optionally
                    preceded by a userID), a question mark, or a
                    question mark preceded by a userID. For example,

                        STATUS MY_TAPE
                        STATUS WABC:MY_TAPE
                        STATUS ?
                        STATUS WABC:?

                    "options" may be specified to produce specific
                    information or to control the output format or
                    destination.   "options" may be one or more of the
                    following:

                    To select specific items:

                        ACCESS   COMMENT   CREATE DATE
                        DENSITY (or MODE)   DRIVES   IDLE DAYS
                        LASTCHG   LAST MOUNTED   LASTREF

| LAST USER   LOCATION   MEDIA
| MOUNT RESULT   NAME   OWNER
| PDN   RACKNUMBER   RECEIPTS
| REFERENCES   RING STATUS   SITE
| STATUS   TAPE NAME   VOLUME

     To select groups of items:

     SHORTINFO   TOTAL

     To control formatting:

     COLUMNS   HEADING   INDENT   KEYWORD
     LABELLED   NOHEADING   PACKED   SPACING
     TERSE   VERBOSE

     To change the output destination:

     OUTPUT=FDname

For example, to display all information about all tapes on *PRINT*, enter

     STATUS ? TOTAL OUTPUT=*PRINT*

To display the last time the tape F000:F000T1 was mounted with write-protection disabled, enter

     STATUS F000:F000T1 LASTCHG

| <u>PER</u>MIT tapename [access [accessor]]
| <u>PER</u>MIT tapename1 LIKE tapename2

The PERMIT command allows the user to permit a tape in a manner similar to the MTS $PERMIT command. The syntax takes the same form as well with the following differences.

When permitting a tape "LIKE" another tape, only one tape name may appear after "LIKE". For example,

     PERMIT TAPE4 LIKE TAPE1

Certain access types are not valid with tapes. The valid access types are

     <u>WR</u>ITE   RENAME (or RNA)   RUN
     RW   <u>R</u>EAD   EDIT
     <u>DEF</u>AULT   <u>D</u>ESTROY   UNLIM
     PERMIT   FULL   <u>N</u>ONE

For example, to permit TAPE1 READ to user WXYZ, enter

```
                    PERMIT TAPE1 R WXYZ
```

Accessors may be the same as for files (userIDs, projectIDs, program keys, or qualified program keys), e.g.:

```
    ALL
    ME
    OTHERS
    OWNERS
    userID
    PROJECT=projectID
    PKEY=pkey
```

| For further information, see Tutorial T7012, "Submitting, Retrieving, and Permitting Magnetic Tapes."

RENAME oldtapename [TO] newtapename

The RENAME command takes the same form as the MTS $RENAME command with one difference, confirmation is not allowed. For example to rename TAPE1 to be POPULATION.DATA, enter

```
    RENAME TAPE1 TO POPULATION.DATA
```

Note: The RENAME command may also be used to
| transfer ownership of a tape to another user just
| like the MTS $RENAME command.

MODIFY tapename option

The MODIFY command changes the pseudodevice name (PDN) for a tape (the default is *T*). For example,

```
    MODIFY TAPE1 PDN=*OUT*
```

The MODIFY command also can change any comments associated with the tape. For example,

```
    MODIFY TAPE2 COMMENTS="Includes data from TAPE1"
```

The comments must be enclosed in quotation marks and may not be more than 100 characters in length.

STOP

The STOP command terminates execution of *TAPES.

MTS

The MTS command returns control to MTS command mode. $RESTART can be used to resume execution of this program.

*TAPESUBMIT/*TAPERETRIEVE

Contents: The magnetic-tape submission and retrieval programs.

Purpose: To guide a magnetic-tape user through the tape-submission
                and retrieval process used at the Computing Center.

Use: The programs are invoked by the $RUN command.

Program Keys: *TAPESUBMIT
                *TAPERETRIEV

Logical I/O Units Referenced:
                SERCOM - error comments, prompting messages, and summary
                         information.
                GUSER - user responses to prompting messages (if any).

Return Codes: Always zero.

Description: TAPESUBMIT is the program that must be run by all
                magnetic-tape users who wish to submit magnetic tapes to
                the Computing Center for subsequent mounting and use.
                Depending on the options selected, the user will supply
                and the program will collect all of the information
                necessary for a magnetic tape to be initialized (option-
                al) and placed in the tape racks at the Computing Center.

                The following commands may be used with TAPESUBMIT.
                Single commands may also be entered in the PAR field of
                the $RUN command.   In the descriptions that follow,
                allowable abbreviations are indicated by underlining.

| DELETE receipt-code
|
| The DELETE option deletes an entry for a tape
| that is in the submission process.

                    DISPLAY

                        The DISPLAY option prints information about the
                        tapes for which the user has run TAPESUBMIT, but
                        have not yet been entered into the tape database.
                        Normally, the information is entered when the
                        tape is presented to Computing Center personnel.

                    HELP topic

                        The HELP command gives detailed information about
                        the requested topic or about obtaining more help
                        if no topic is given. Possible topics include:

        any other TAPESUBMIT commands, PROCEDURES, GLOS-
        SARY, BULKINPUT, or EXAMPLES.

MTS

    The MTS command returns control to MTS command
    mode. TAPESUBMIT may be re-entered with the MTS
    $RESTART command.

SUBMIT (NEW|OLD|TERSE|VERBOSE)
SUBMIT (INPUT=filename)

    The SUBMIT command initiates the tape-submission
    process and has two forms.

    The first form above is used to submit a small
    number of tapes. The user will be asked and
    should respond to a number of questions concern-
    ing the tape to be submitted. By specifying the
    NEW or OLD option, a few questions necessary to
    determine the status of the tape will be skipped,
    shortening the process. If the tape has been
    used prior to this submission but there is no
    further need to access the data on the tape, then
    NEW may be specified or both NEW and OLD may be
    omitted. Once the user has become familiar with
    the questions and possible answers, the TERSE
    option may be specified to eliminate the detailed
    questions. Only use the TERSE option after first
    running through a submission without the option.

    The second form above is used to submit a large
    number of tapes. For this method the information
    about each tape is contained on a single line in
    a file. The INPUT keyword of the command speci-
    fies the name of the file. All lines in the file
    that do not properly describe a tape will be
    ignored and processing will continue with the
    next line.

    Information collected using the two forms of this
    command will be recorded for future use when the
    tapes are actually delivered to the Computing
    Center operations staff.

STOP

    The STOP command terminates execution of the
    program.

          $mts-command

               Any command beginning with a dollar sign ($) is
               interpreted as an MTS command.

     TAPERETRIEVE is the program that must be run when a user
     wishes to retrieve a magnetic tape that has been previ-
     ously submitted to the Computing Center.   The only
     parameter required by TAPERETRIEVE is the tape name,
     which is specified in the PAR field of the $RUN command,
     e.g.,

          $RUN *TAPERETRIEVE PAR=tape-name

     For further information on the tape-submission and re-
     trieval procedures, see Computing Center Memo 471, "Sub-
     mitting, Retrieving, and Permitting Magnetic Tapes."

January 1987

<u>*TAXIR</u>

Contents: The Taxir information retrieval system.

Use: The program is invoked by the $RUN command or may be
                 called as a subroutine from a user program.

Program Key: *TAXIR

Logical I/O Units Referenced:

                 None.

Return Codes: Always zero.

Description: Taxir is a general-purpose information storage and re-
                 trieval system with facilities for building, updating,
                 and querying user-defined data banks. Taxir includes
                 features for generating reports that may include head-
                 ings, labels, n-way tabulations, totals, and subtotals.
                 Taxir also provides an interface to the MIDAS statistical
                 analysis system (MIDAS is supported by the U of M
                 Statistical Research Laboratory).

                 For the complete description of Taxir, see <u>The Taxir
                 Primer</u>, 4th edition, by R. C. Brill. This edition de-
                 scribes the same version of Taxir as the 3rd edition with
                 Updates 1 and 2 applied. It differs from the 3rd edition
                 in the following particulars:

                 (1) The contents of Update 1 (CCMemo 418) and Update
                      2 (CCMemo 433) have been integrated into the
                      text.
                 (2) Some parts of the text, notably the examples,
                      have been modified to reflect changes introduced
                      by the updates.
                 (3) The example data bank in Part I, now appearing in
                      Chapter 2, has been extensively revised and
                      expanded.
                 (4) A new section describing common errors in Boolean
                      expressions has been added to Chapter 8.
                 (5) Obsolete features that are still available in the
                      system have been removed from the main text and
                      have been collected together as Appendix B.
                 (6) A new section, Appendix A, discusses design
                      strategies for fitting data into Taxir's flat-
                      file data structure.

<u>*TDAY88TRANS</u>

Contents: A server program to translate files to use the IBM EBCDIC
                coding conventions.

Use: The program is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:
                SCARDS - input command lines for the translation program.
                SPRINT - output messages from the translation program.

Parameters: A single command line may be entered in the PAR field of
                the $RUN command.

Return Codes: Always zero

Description: On Translation Day, February 22, 1988, the Computing
                Center changed the codes used to represent some charac-
                ters on MTS. As more and more people transfer informa-
                tion between microcomputers and mainframes and between
                mainframes and other machines, the translation of some of
                the less commonly used characters becomes more of an
                annoyance. The changes made on T-Day improve the ability
                of those using MTS to move files or send messages to
                other hosts easily without having characters become
                unreadable or having to translate a file that contains
                them upon arrival at another host.

                This change brings MTS in line with the current Interna-
                tional Standards Organization (ISO) character code stand-
                ard and the newer EBCDIC standard based on this ISO
                standard.

                For three main reasons, the Computing Center is providing
                a free Translation Server on UM-MTS and UB-MTS to help
                users convert files affected by T-Day:

                    (1) To give users a quick and easy way to find out if
                         a file has the affected characters in it.

                    (2) To save users money when they translate files to
                         the new codes.

                    (3) To help users by issuing a warning if the file
                         seems like one that should not be translated, or
                         if the file has already been translated.

The server is also designed to preserve file security and provide a simple and error-free mode of operation.

To access the MTS translation server, you must run an interface program on your CCID to translate files on that CCID only. Users with several CCIDs will have to translate the files on each one in a separate signon. To run the server, enter the command

     $RUN *TDAY88TRANS

When the server prompts for a command, enter one of the following:

     SCAN file1,file2,...,filen
     TRANSLATE file1,file2,...,filen
     STOP

where "file1", "file2", etc., indicate one or more file names separated by commas or blanks. To run the server with a single command that scans a file or list of files for the affected characters, enter the command

     $RUN *TDAY88TRANS PAR=SCAN file1,file2,...,filen

To run the server with a single command to translate a file or files, enter the command

     $RUN *TDAY88TRANS PAR=TRANSLATE file1,file2,...,filen

with a file name or list of files as described above.

Since the server is run from your own CCID, you do not have to explicitly permit files to the server. This prevents an extra nuisance step, and also avoids file-security risks from requiring users to permit their files to the server. This means, however, that the program can only change the files on that CCID. Even files on another CCID to which your current CCID has UNLIMITED access will not be translatable from your current CCID.

The server can be run from a terminal or in batch mode.

*TDAY88TRANS validates the file (ensures the file is indeed owned by the user), permits the file to the server, and then turns control over to the server. The server then performs the actual diagnostics and character translation. Upon returning from the server, *TDAY88-TRANS depermits the file to the server and asks for another file or files to convert. Because the interface program is running on your CCID, there is a minimal cost for the work it does and for the connect time, but it is

less expensive than using the MTS Editor for small files and a considerable savings for large files.

The server will warn you, and refuse to translate the file, if it finds that it contains any non-printing characters. This will cause it to reject binary code that should not be translated, such as object code, ILIR:MICRO and STAT:MIDAS internal files, SPSSx internal files, Full-Screen Message archive files, and so on. It will warn about page printer-ready files containing $9700 lines, since there is no way it can tell if those should be translated or not.

The server will also print a warning and refuse to translate the file if it has already translated it. It keeps track in its own file database, a method that is not infallible. For example, if a file was translated using the MTS Editor, the server has no way of knowing that file should not be translated. It also will not know if a file it has recorded as translated has been destroyed and then restored off tape in an untranslated version, and therefore truly needs translation, even though it is on the list of already-translated files. If you rename a file after it has been translated by the server, you might translate it twice by mistake. Be cautious: if you have any doubts, look at the file first to see if affected characters have been translated.

The server runs only on whole files. It does not allow filename patterns, $CONTINUE WITH lines, or explicit concatenation of files, e.g., a+b+c+d. If you have a file that contains mixed text and binary data, such as a program file with the source code at one line range and the object at another, you will need to use the MTS Editor to translate the line ranges or individual lines that need changing. In general, if the server will not translate a file for some reason, you can use the Editor instead. Through the normal rebate procedures, you can obtain rebates for translating files using the Editor.

The translation server runs on one host at a time, and does not allow cross-host translations. This means that if you have files on UM and UB you will need to sign on to each system separately to run the server. Again, this simplifies file security issues.

If there is a system crash, or some other interruption in service while you are running the server, the program might be in the middle of translating a file and be unable to complete the translation. This could leave you with a damaged file. Therefore, before the server begins to translate a file it will depermit it so you do not have access to it. It will only restore the original

file access once the translation is completed. This means you can tell if a file being translated was damaged after a crash; MTS will tell you "Access not allowed". Use *RESTORE to restore a correct, untranslated version, and use the $PERMIT command to get access to the file again.

See Computing Center Memo 480, "Translation Day Character Code Changes in MTS," for further information on translating MTS files.

January 1987

TELLABANK

Contents: The TELLABANK program.

Use: The program is invoked by the $RUN command.

Program Key: *TELLABANK

Logical I/O Units Referenced:
                 3   - vocabulary file; defaults to (and must be)
                         TGRF:TAGSYB4.0.
                 4   - bank data file; defaults to TAGPRM if it exists,
                         otherwise it defaults to -TAGPRM.
                 5   - terminal input, equated to SCARDS.
                 6   - printed terminal output.
                 8   - raw data input file; prompted for (do not
                         assign).
                 9   - temporary file -PRMEDIT.
                 11   - temporary file -TAGSCR23.
                 13   - bank data copy file -PRMCOPY.

Return Codes: Always zero.

Description: TELLABANK serves as an alternate means of entering,
                deleting, and editing data for use by TELL-A-GRAF.
                TELLABANK, unlike TELL-A-GRAF, allows input of

                    (1) more than two variables at a time,
                    (2) lines longer than 72 characters, and
                    (3) FORTRAN-formatted data.

                A PRM file is a specially formatted file used by
                TELL-A-GRAF to retrieve data for plotting. This file is
                also used to store and retrieve plots using SUBPLOT and
                DRAW. Data is retrieved from the PRM file by using the
                TELL-A-GRAF BANK DATA command. TELLABANK automatically
                enters data into the PRM file. TELLABANK can also delete
                and edit this data. Once the data has been entered into
                the PRM file, TELL-A-GRAF can be used to plot it.

                Further information on using TELLABANK is available in
                Computing Center Memo 450, "TELL-A-GRAF in MTS," or by
                copying the file TGRF:TELLAGRAF.W.

442   *TELLABANK

January 1987

*TELLAGRAF

Contents: The Tell-A-Graf conversational graphics program.

Use: The program is invoked by the $RUN command.

Program Key: *TELLAGRAF

Logical I/O Units Referenced:
                SCARDS - CONTROL file; defaults to *SOURCE*.
                SPUNCH - SAVE file; defaults to -TAGSAVE(*L+1).
                SPRINT - TRACE file; defaults to -TAGTRA or *SINK*.
                0   - POP output; no default.
                3   - vocabulary file; defaults to (and must be)
                         TGRF:TAGSYB4.0.
                4   - bank data file; defaults to TAGPRM if it exists,
                         otherwise defaults to -TAGPRM.
                5   - terminal input; assigned to SCARDS.
                6   - printed terminal output.
                8   - profile file; defaults to TAGPRO if it exists,
                         otherwise defaults to an appropriate file under
                         TGRF ID.
                9   - Calcomp plot file output; no default.
                11   - temporary file -TAGSCR23.
                13   - bank data copy file -PRMCOPY.
                14   - ISSCO's SLIDES output; no default.

Return Codes: Always zero.

Description: TELL-A-GRAF, a proprietary software product of ISSCO of
                San Diego, is a stand-alone program for doing two-
                dimensional computer graphics.   TELL-A-GRAF is device-
                independent and hence can be run with a variety of output
                devices. It has high-quality output, and its command
                language is designed to be understood by those with
                little computing experience.

                TELL-A-GRAF is documented extensively by the "TELL-A-GRAF
                User's Manual" published by ISSCO. Copies may be pur-
                chased in the local bookstores and are available for
                reference in the documentation racks at the public batch
                stations.

                TELL-A-GRAF is a conversational program for producing
                high-quality two-dimensional graphs, bar charts, pie
                charts, and histograms. The user has a wide variety of
                character fonts, axis types, shading patterns, and other
                enhancements to choose from, all of which may be allowed
                to assume simple default values.

Because of contractual restrictions, TELL-A-GRAF is sur-charged when run from an industrial (commercial) ID. The surcharge is to cover royalties to ISSCO that are required if TELL-A-GRAF is used by non-university users.

Further information on using TELL-A-GRAF in MTS is available in Computing Center Memo 450, "TELL-A-GRAF in MTS," or by copying the file TGRF:TELLAGRAF.W.

Parameter: PAR=TRACE may be specified in the PAR field of the $RUN command in order to keep a trace record of a TELL-A-GRAF session. By default, TRACE is enabled in batch mode, and disabled in terminal mode. If enabled, output will be sent to the printer (*SINK*) in batch mode, or to -TAGTRA in terminal mode. TRACE output may be routed to another destination by assigning SPRINT on the $RUN command, e.g.,

           $RUN *TELLAGRAF SPRINT=*PRINT* PAR=TRACE

will send a session record to the printer. PAR=NOTRACE disables tracing.

<u>*TERMLIST</u>

Contents: A list of terminals that have been thoroughly checked out
                or used on MTS via the dial-up telephone lines.

Use: The list may be obtained via the $COPY command, e.g.,

                   $COPY *TERMLIST *PRINT*

*TERMTYPES

Contents: A list of the ASCII terminals that are supported by the
               UMnet Computer Network. This list includes the terminal-
               type code, the default screen width, the carriage-return
               timing, and settings of other features.

Use: The list may be obtained via the $COPY command, e.g.,

                    $COPY *TERMTYPES *PRINT*

January 1987

*TEX

Contents: The TeX text-processing program.

Use: TeX is invoked by the $RUN command.

Program Key: *TEX

Logical I/O Units Referenced:
                SERCOM - error messages, error recovery prompting, and
                         diagnostics.
                GUSER - error recovery prompt input.

                TeX references all files explicitly by name.

Return Codes: Always zero.

Description: The complete description of the TeX text-processing
                language is given in the following publications:

                    The TeXbook by Don Knuth, Addison-Wesley Publishing
                        Company (1984), ISBN 0-201-13448-9.
                    Computing Center Memo 810, "An Introduction to TeX."

                TeX is an extremely versatile text-processing language
                intended for use with high-quality output devices such as
                a phototypesetter.   Currently, the only device at the
                University of Michigan that can approximate TeX's capabi-
                lities on MTS is the Xerox 9700 page printer.

                Mathematics and technical applications that use special
                characters or accents are TeX's strong point; other text
                processors may be more appropriate for simple documents.
                TeX is device-independent, which means that commands used
                to produce a document on one device need not be changed
                to produce the same document on another device.

                TeX accepts an input file containing text to be formatted
                and control codes to direct the formatting process and
                produces a device-independent (DVI) output file. The DVI
                file is written in internal (binary) format. This file
                must be processed by another program to produce readable
                hard-copy results.   The program *DVIXER transforms the
                DVI file into one that can be sent to the Xerox 9700 and
                printed by running *PAGEPR as the final step. Support
                for use of TeX with a phototypesetter is also available.

                TeX is designed principally for the production of multi-
                page documents. Although TeX is quite capable of produc-
                ing small documents, the cost associated with initializa-

tion of TeX may be more than the cost of formatting a
small document.

January 1987

*TEXTFORM

Contents: The TEXTFORM text-processing program.

Use: TEXTFORM is invoked by the $RUN command.

Program Key: *TEXTFORM

Logical I/O Units Referenced:
               SCARDS - text input to TEXTFORM.
               SPRINT - output listings produced by TEXTFORM.
               SPUNCH - formatted text produced by TEXTFORM.
               SERCOM - error messages and diagnostics.

Return Codes: 0 - Successful return.
               >0 - Return code is highest severity level that occurred
                    during execution. From 4 to 16, the document was
                    completed; for 20 and higher, the document is
                    incomplete.

Description: The complete description of the TEXTFORM text-processing
               language is given in the Textfornm Reference Manual.
               This manual is based on the manual TEXTFORM, published by
               the University of Alberta (1984).   An introductory de-
               scription of TEXTFORM is given in Computing Center Memo
               820, "Introduction to TEXTFORM"; a TEXTFORM command
               reference summary is given in Computing Center Memo 823,
               "TEXTFORM Reference Summary."

               TEXTFORM can produce a variety of documents ranging from
               single-page memos and letters to complete manuscripts.
               The language is device independent, which means that
               commands used to produce a document on one device need
               not be changed to produce the same document on another
               device.   Although the output may be slightly different,
               depending upon the capabilities of the intended output
               device, the general form of the document will remain
               unchanged.

               TEXTFORM is designed principally for the production of
               multi-page documents. Although TEXTFORM is quite capable
               of producing small documents, the cost associated with
               initialization of TEXTFORM may be more than the cost of
               formatting a small document.

               The main strength of TEXTFORM is its power to drive
               sophisticated output devices.   Although TEXTFORM can
               drive simple devices such as the line printer, its
               features and facilities were designed for more advanced
               devices such as the Xerox 9700 and various phototypeset-

ters. As a result, TEXTFORM may seem rather unwieldy and expensive for simple devices.

TEXTFORM may be used with the Xerox 9700 page printer, the line printer, and the APS Micro 5 phototypesetter.

January 1987

<u>*TYPEQ</u>

Contents: The program that queues files to be copied to the
               phototypesetter.

Purpose: To place the user's typesetting job(s) on the queue for
               typesetting.

Use: This program is invoked by the $RUN command.

Program Key: *TYPEQ

Logical I/O Units Referenced:
               SCARDS - queueing requests.
               SPRINT - job receipt numbers and prompting messages for
                        queueing requests.
               SERCOM - error comments.
               GUSER - user responses to error comments.

Parameters: A single queueing request may be specified in the PAR
               field of the $RUN command, if no requests are to be read
               from SCARDS.

Return Codes: Always zero.

Description: Each queueing request specifies a file or tape containing
               a typesetting job (usually the output from a text-
               processor) and, optionally, the width of the paper to be
               used and a delivery code. *TYPEQ reads the job, deter-
               mines the cost of typesetting, places the request in the
               system typesetting queue, and charges the user for the
               typesetting costs.

               *TYPEQ prompts for queueing requests using the message

                   Enter job request (and delivery code):

               The response may be a queueing request for a file or
               tape, a line beginning with a "$", or an end-of-file
               indicator.   If the queueing request is for a file, the
               request should consist of the file name, and optionally
               followed by the paper width and/or the delivery parame-
               ters. Explicit file concatenations are permitted, as are
               file line-number ranges. Only permanent files may be
               queued--temporary files may not be queued.

               A response that begins with a dollar sign is treated as
               an MTS command that is to be executed immediately.
               Execution of *TYPEQ may be resumed by entering a $RESTART
               command (unless an MTS command causes *TYPEQ to be

unloaded). An end-of-file response terminates execution
of *TYPEQ.

If the request is for a magnetic tape, the tape should
already be mounted. The request should contain first the
pseudodevice name, followed by the tape ID.   Parameters
must be separated by one or more blanks and/or commas.
There are three optional parameters. One is the paper
width, which may appear anywhere after the pseudodevice
name. Another is the FILES parameter, which may also
appear anywhere after the pseudodevice name. This takes
the form FILES=n, where "n" is the number of files to be
typeset.   The default is FILES=1.   Finally, the POSN
parameter may appear anywhere after the tape ID.   This
takes the form, POSN=string, where "string" specifies a
file on the tape (any string legal for the POSN control
command).   If this parameter is not specified, *TYPEQ
begins reading at the current tape position (even if that
is in the middle of a file).   (At postprocessing, the
tape will be mounted and positioned appropriately, even
to the middle of a file.) *TYPEQ and the postprocessor
use whatever blocking is in effect at the time the
queueing request is entered. *TYPEQ does not rewind the
tape after reading it. Pool tapes may not be queued.

A job request may specify the width of the paper
(typesetting media) to be used. This parameter is given
in the form

     WIDTH=n

where "n" is the width of the paper in inches (either 6,
8, or 12). If the width specified is not available, the
next larger available size is selected. If this parame-
ter is omitted, 8-inch paper is used.

Users can request delivery of output to another station
by specifying the DELIVERY=station parameter on the queue
request, e.g., DELIVERY=NUBS. The DELIVERY parameter is
effective for only one queue request. If the DELIVERY
parameter is not specified, the setting of the MTS $SET
DELIVERY option will be used (which defaults to the
Computing Center).

*TYPEQ checks for errors such as illegal commands and
requests for non-existent fonts as it reads the job.   If
errors are detected, *TYPEQ prints the number of errors
discovered. The job may be queued with errors present,
but this is not recommended.

A line beginning with $CONTINUE WITH ... will be treated
as an invalid input. Implicit concatenation lines may
not be used in the job file.

Queueing:

After *TYPEQ finishes reading the job, it prints the
number of errors in the job (if any), the number of
phototypesetter units (ptus) required, the amount of
paper required, and the cost. In conversational mode,
the user is asked whether the job should actually be
queued. If the answer is NO, the job is aborted and the
user is prompted for a new job request (unless the job
request was taken from the PAR field). If the answer is
YES, or if *TYPEQ is being executed in batch mode, the
job request is placed in the queue, the user is charged
for the job, and a receipt number is printed (this
receipt number must be used to obtain the output at the
output window later). In addition, if the job input was
contained in one or more disk files, *TYPEQ will, if
necessary, permit the queued files so that the postpro-
cessor, which is a program with PKEY=*TYPEQ, may access
the job input. *TYPEQ will establish READ access for the
file to the postprocessor, if necessary. (Specifically,
the file will be permitted "READ PKEY=*TYPEQ" unless the
postprocessor already has read access. See the descrip-
tion of the $PERMIT command in MTS Volume 1, Michigan
Terminal System.) A message is printed for each file so
permitted. If *TYPEQ cannot determine the permit status
of a file, a message is printed, and the user must be
sure that the file is permitted before the job request is
postprocessed (see the section below on postprocessing).

Checking the Status of a Job

The status of a queued job may be determined by entering
the command

      QUEUE nnnnnn

where "nnnnnn" is the job receipt number. This can only
be done by the signon ID that queued the job. *TYPEQ
prints a message indicating whether the job is waiting to
be typeset, has been typeset but is waiting to be saved
on tape, has been canceled, or is not in the queue. Once
a job has been saved on tape, it is removed from the
queue; hence, no information is available after the job
is saved.

Cancellation:

A queued job may be canceled by entering

      CANCEL nnnnnn

where "nnnnnn" is the job receipt number, whenever *TYPEQ
is prompting for a job request. A job can be canceled

only by the signon ID which queued it. If the job specified has not been typeset, it will be canceled and a message to that effect will be printed. Typesetting charges will be rebated automatically at a later time.

Attention Interrupt Processing:

If the user issues an attention interrupt, processing of the current job request is suspended; the subsequent action taken by *TYPEQ is as follows: (1) If it was parsing a job request, the job request is discarded and the user is prompted to enter a new request; (2) *TYPEQ will print "READING:" or "QUEUEING:", depending upon whether it was in the process of reading the job input or queueing the job. In either case, if the user enters a null line or an end-of-file, *TYPEQ will resume processing the current request. If the user enters "MTS" or an MTS command beginning with a "$", *TYPEQ will return to MTS command mode; if a command was given, it will be immediately executed by MTS. If the user enters anything else, the job request will be aborted.

If the user issues a second attention interrupt while *TYPEQ is processing the first interrupt, an immediate return is made to MTS command mode. The user may subsequently re-enter *TYPEQ via the $RESTART command.

Postprocessing:

After a job request has been queued, the user's job input must be available for reading by the system postprocessor as soon as the typesetter is available and at the scheduled job saving time. If the job input is in a file, the file must be permitted, and it must not be locked at any level higher than READ. If the file is not permitted, the job will be canceled; if it is locked, the job request will remain queued until it is accessible. If the job input is on a tape, the tape must be available for mounting (e.g., not already mounted); otherwise, the job will remain queued until the tape is available. If the tape ID is incorrect, the job will be canceled.

Examples: In the following examples, terminal output appears in uppercase and user input appears in lowercase.

```
#r *typeq
#EXECUTION BEGINS
 ENTER JOB REQUEST (AND DELIVERY CODE):
 typefile1 w=12
 JOB REQUIRES  .2 SQ.FT.;  $.46
 OK?
 no
 ENTER JOB REQUEST (AND DELIVERY CODE):
```

```
            typefile2 width=8
            JOB REQUIRES  .1 SQ.FT.;  $.81
            OK?
            ok
            JOB ASSIGNED RECEIPT # 790397, DUMWAIT3
             **LEAVE THE JOB FILE INTACT THROUGH THE NEXT JOB
               COLLECTION TIME.
            ENTER JOB REQUEST (AND DELIVERY CODE):
            typefile3 w=12 delivery=nubs
            JOB REQUIRES  .3 SQ.FT.;  $2.52
            OK?
            ok
            JOB ASSIGNED RECEIPT # 790398, NUBS
             **LEAVE THE JOB FILE INTACT THROUGH THE NEXT JOB
               COLLECTION TIME.
            ENTER JOB REQUEST (AND DELIVERY CODE):
            chapter1 w=6
            JOB REQUIRES  .1 SQ.FT.;  $1.15
            OK?
            ok
            JOB ASSIGNED RECEIPT # 790399, DUMWAIT3
             **LEAVE THE JOB FILE INTACT THROUGH THE NEXT JOB
               COLLECTION TIME.
            ENTER JOB REQUEST (AND DELIVERY CODE):
            queue 790399
            790399 IS WAITING TO BE RUN.
            ENTER JOB REQUEST (AND DELIVERY CODE):
            cancel 790399
            790399 HAS BEEN CANCELLED.
            ENTER JOB REQUEST (AND DELIVERY CODE):
            {end-of-file}
           #EXECUTION TERMINATED
```

January 1987

*UNEDIT

Contents: The program to unedit a line file.

Purpose: To compare a current version   of a line file with the
                 original version and produce context editor control cards
                 for transforming the original version into   the   current
                 version.

Use: The program is invoked by the $RUN command.

Program Key: *UNEDIT

Logical I/O Units Referenced:
                 GUSER - prompting responses.
                 SERCOM - prompting messages for   the   FDnames and error
                           comments.
                 SPUNCH - context editor control commands produced.
                 0   - original version of the file.
                 1   - current version of the file.
                 SPRINT - PAR=LIST output

Parameters: The following parameters may be specified   in   the   PAR
                 field of the $RUN command.   The parameters   must be
                 separated by commas.

                 MAXDEL=n   "n" is an estimate of the maximum number of
                            lines to be deleted in any one deletion from
                            the original version of the file.   Specifying
                            a value slightly larger   than the number of
                            lines actually deleted will greatly reduce the
                            cost of running the program. If "n" is too
                            small, the output written to SPUNCH will be
                            valid, but much longer than necessary.   The
                            default is plus-infinity.

                 MAXCOMP=n "n" is the maximum number of lines to be
                            compared between the original and current
                            versions before deciding that a "resynch" has
                            been found. The actual number of lines com-
                            pared depends on the size of the preceding
                            deletion and the value of the BREAK and SLOPE
                            parameters. The default is 25.

                 BREAK=n   "n" and "m" control the number of lines   that
                 SLOPE=m   must match between the original and current
                            files before a "resynch" is recognized.   If
                            "d" is the size in lines of the deletion
                            preceding the resynch, then the match must be
                            "n+(d-n)/m" lines long, unless "d<n", in which

case "d" is used. If the result is greater
than the value of MAXCOMP, then MAXCOMP is
used. The defaults are n=3 and m=7.

LIST   A listing of the differences between the old
and new versions will be printed on SPRINT.

Return Codes: 0 - Successful return.
4 - Error return.

Description: The *UNEDIT program reads two versions of a file: the
original version and the current version. Both files
must be _line_ files. The entire contents of each line
(including trailing blanks) is compared. The line num-
bers are used in the editor control commands output, but
are ignored during the comparison process. The output
from *UNEDIT consists of a set of context editor control
commands which will transform the original version of the
file into the current version. The effect of applying
the output of *UNEDIT as commands to the context editor
working with the original file will produce, as a result,
a file with lines that have the same content as the
current file and in the same order, but not necessarily
with the same line numbers.

If units 0 and/or 1 are not assigned, the program will
prompt for the FDnames of the two files via the logical
I/O unit SERCOM; the FDnames of the two files will be
read from GUSER.

For a description of the context editor, see the "Edit
Mode" section of the "System Command Language" in MTS
Volume 1, _Michigan Terminal System_. Two other programs,
*DOWNDATE and *APC, are also available for comparing two
versions of a file and producing the changes. A discus-
sion of the differences of these three programs is given
in the "Files and Devices" section of MTS Volume 1.

Example: $RUN *UNEDIT 0=OLDFILE 1=NEWFILE SPUNCH=CHANGES

In the above example, the original file OLDFILE is
compared with the current file NEWFILE. The context
editor control commands are written to the file
CHANGES.

January 1987

USERDIRECTORY

Contents: A program that maintains a directory of names and
                 associated information.

Use: The program is invoked by the $RUN command.

Program Key: *USERDIRECTO

Logical I/O Units Referenced:
                 SCARDS - input command to User Directory.
                 SPRINT - printed output from User Directory.
                 SERCOM - error messages, prompting messages, and help
                          text.
                 GUSER - responses to prompting messages.

Return Codes: Always zero.

Description: The User Directory program maintains a database, called
                 the master directory, that allows names to be used with
                 the MTS Message System.

                 There are four types of name entries that can be put into
                 a directory, individual names, remote names, group names,
                 and alias names.

                 The most common is the individual name.   Messages that
                 are sent to an individual name are put in the mailbox of
                 the owner of that name.

                 The second type of name entry is the group name. When a
                 group name is added to the directory, the user will be
                 asked to give a list of "members", which may be a
                 combination of individual names, remote names, userIDs,
                 and projectIDs. Then, whenever a message is sent to this
                 group name, it will be sent to each of the members and to
                 the owner of the name. For example, if "Pascal Language
                 Project" is a group with members Marlou Smith, John
                 Brown, and Jack Black, and a message is sent to this
                 name, the message will be sent to Marlou Smith, John
                 Brown, and Jack Black. Groups are considered to be in
                 the public domain, that is, any user may send a message
                 to any group that is enrolled in the master directory.

                 The third type of name entry is the alias name. This can
                 only be used with name libraries. When an alias name is
                 enrolled in the name library, the user will be asked to
                 give a "primary name", which must be a name that exists
                 in the master directory. The primary name may be either
                 an individual name, a remote name, or a group name.

Whenever a message is sent to the alias name, it will appear that the message was sent to the primary name. Name libraries are considered to be private, that is, only a user that has access to the name-library file may send a message to individual participants (using the alias names) or to a group in the name library.

The fourth type of name entry is the remote mail name. Messages that are sent to a remote mail name are put into the remote mailbox of the name. For example, sending a message to John Q. Public with the remote mail name JQPublic@MIT is sent to the site MIT and then delivered to the user JQPublic.

Instructions for enrolling individual names is given below. Instructions for enrolling group names, alias names with name libraries, and remote mail names is given in MTS Volume 23, <u>Messaging and Conferencing in MTS</u>.

Each individual name first must be enrolled in the master directory. The User Directory program prompts the user for information about his or her own name and how it is to be used. The following terminal example illustrates how an individual name may be enrolled (some of the explanatory information printed by the program is omitted for the sake of brevity).

```
#$run *userdirectory
#Execution begins

 Whenever you need help, type "HELP".

*add

 Enter name (alternate spellings in parentheses) or
 "HELP":
 Example:  Doug (Douglas) R. (Robert, Bob) Hofstadter
?Mary (Marlou) Louise Smith

 Enter business phone (maximum 63 characters) or
 "HELP":
 Example:  313-764-2121
?313-764-9595

 Enter business address (up to 5 lines, 63 characters
 per line), or "HELP".  Hit return when done.
 Example:  Computer Science Department
           221 Angell Hall
           University of Michigan
           Ann Arbor, MI 48109
           USA
1?Computing Center
```

```
            2?University of Michigan
            3?Ann Arbor, MI 48109
            4?{return}
             "Mary Louise Smith" was added with owner WABC
            *display 'mary louise smith'
             "Mary (Marlou) Louise Smith", Type=Individual,
             Owner=WABC
               Fullaccess=(Unlimited Owner, Read Others)
               Busphone:       313-764-9595
               Busaddress:     Computing Center
                               University of Michigan
                               Ann Arbor, MI 48109
            *stop
            #Execution terminated
```

The User Directory program first prompts for the user's complete name in the order first name, middle name, and last name. Alternate names or nicknames may be given in parentheses as illustrated above. After the name has been entered, the user is prompted for information associated with the name, such as a business telephone number and business address. These items may be omitted by entering a null line.

The current userID is taken as the owner userID. This is the ID to which all messages will be directed for this name, e.g., all messages sent to Mary Louise Smith will go to the mailbox of userID WABC.

When a message is sent to a name, any combination of given names or alternate names may be used. For example, the message may be sent to Mary Louise Smith, Mary Smith, Louise Smith, or Marlou Smith. If the name is similar to another name in the directory, the sender will be prompted further to resolve the name. When the message system lists the sender of a message, the complete given name (excluding alternate names) is used.

The DISPLAY command may be used to display information about an entry in the database. The DISPLAY command is normally given as:

        DISPLAY 'name'

For example, either of the following commands will display information about Mary Louise Smith (either given names or alternate names may be used):

        DISPLAY 'Mary Louise Smith'

The primes are not required if the name is given without any internal blanks or parentheses; the underscore may be used to replace blanks, e.g.,

                    DISPLAY Marlou_Smith

                    The DISPLAY command allows the use of the User Directory
                    database as an on-line directory.   Information may be
                    displayed about any person or member of a group in the
                    User Directory database.

Commands: A brief summary of the User Directory commands is given
                   below.   Abbreviations are indicated by underlining.   The
                   complete description of these commands is given in MTS
                   Volume 23.

                   ADD [{GROUP|REMOTE}] [parameter ...]

                        Parameters:

                            [NAME=]'name'
                            MEMBERS={ID=ccid|P=projid}
                            MEMBERS={'name'|('name1','name2',...)}
                            PRIMARYNAME='name'
                            BUSPHONE='number'
                            BUSADDRESS={'addr'|('addr1','addr2',...)}
                            HOMEPHONE='number'
                            HOMEADDRESS={'addr'|('addr1','addr2',...)}
                            MISCELLANEOUS='string'
                            MAILNAME='name@destination'
                            REMOTENAME='name'
                            SITE='destination'
                            FULL
                            OWNER=ccid

                   CLAIM {'name'|EACH OWNER=ccid}

| CLEAR group

                   COMMENT text

                   DESTROY identifier [OK]

                   {DISPLAY|FIND} identifier [parameter ...] [OUTPUT=FDname]

                        Identifiers:

                            'name'
                            MEMBER='name'
                            ME or MY

                        Parameters:

                            NAME
                            NAMEID
                            OWNER
                            TYPE

                         MEMBERS
                         PRIMARYNAME
                         MAILNAME
                         REMOTENAME
                         SITE
                         SITEBOX
                         BUSPHONE
                         BUSADDRESS
                         HOMEPHONE
                         HOMEADDRESS
                         MISCELLANEOUS
                         CREATEDATE
                         CREATESIGNONID
                         LASTSETDATE
                         LASTSETSIGNONID
                         LASTMODDATE
                         LASTMODSIGNONID
                         ACCESS
                         FULLACCESS or FA
                         FULL
                         TOTAL

| EMPTY group

               {HELP|EXPLAIN} [parameter]

                    Parameters:

                         INTRODUCTION
                         GETTING STARTED
                         COMMANDS
                         MESSAGESYSTEM
                         NAMELIB
                         CHANGES

               JOIN 'name'

               MCMD MTS‑command

               MODIFY identifier parameter ...

                    Identifiers:

                    'name'
                    MEMBER='name'
                    ME

                    Parameters:

                    NAME='name'
                    MEMBERS={ID=ccid|P=projid}
                    MEMBERS={'name'|('name1','name2',...)}
                    PRIMARYNAME='name'

                                        *USERDIRECTORY   465

                             MAILNAME='name@destination'
                             REMOTENAME='name'
                             SITE='destination'
                             BUSPHONE='number'
                             BUSADDRESS={'addr'|('addr1','addr2',...)}
                             HOMEPHONE='number'
                             HOMEADDRESS={'addr'|('addr1','addr2',...)}
                             MISCELLANEOUS='string'
                             FULL
                             OWNER=ccid

                     MTS [MTS-command]

                     PERMIT identifier access accessor, ...

                             Identifiers:

                                'name'
                                MEMBER='name'
                                ME

                             Accesses:

                                READ
                                SET
                                MODIFY
                                DESTROY
                                JOIN
                                PERMIT
                                FULL
                                UNLIMITED
                                PWSET
                                DEFAULT

                             Accessors:

                                ID=ccid
                                PROJECTID=projid
                                'name'
                                OWNER
                                OTHERS
                                ALL
                                ME

                     RESIGN 'name'

                     RETURN

                     SET option ...

                             BATCH={ON|OFF}
                             BRIEF={ON|OFF}
                             ECHO={ON|OFF}

```
                    HELP=FDname
                    HELPTRACE={FULL|ON|OFF}
                    LOG=logfile
                    MASTER=FDname
                    NAMELIB={FDname|OFF}
                    PARSETRACE={FULL|ON|OFF}
                    PROMPT={ON|OFF}
                    TERSE={ON|OFF}
                    VERBOSE={ON|OFF}
```

| STATUS

               {STOP|QUIT}

               USE {MASTER|NAMELIBRARY [filename]}

*UTILISP

Contents: The University of Tokyo Interactive List Processor.

Use: The processor is invoked by the $RUN command.

Program Key: *UTILISP

Return Codes: n - Argument passed from QUIT function.

Description: The UTILISP (University of Tokyo Interactive LISt Proces-
                 sor) system is designed for highly interactive program-
                 ming and debugging of sophisticated programs.

                 At present, the UTILISP system is in operation at the
                 University of Michigan Computing Center, and is organized
                 as a MTS program.   The standard command sequence for
                 running the system is currently as follows:

                     #RUN *UTILISP PAR=options
                     > (Lisp input)

                 The PAR options are

                     STACK

                         STACK=n specifies that the size of the stack area
                         is to be "n" pages (1 page = 4 kilobytes).   The
                         default value for "n" is 16.

                     FIX

                         FIX=n specifies that the size of the "fixed heap"
                         area (for compiled codes) is to be "n" pages.
                         The default value for "n" is 8.

                     SIZE

                         SIZE=n specifies that "n" pages are to be allo-
                         cated for use by UTILISP. The program will not
                         get more space if this is not enough.   The
                         default value for "n" is 64.

                 If the logical I/O unit 0 has been assigned, the file
                 associated with it is executed first. This execution is
                 identical with that of the standard top-level Lisp loop,
                 except that the results are not displayed.

                 After the execution of the unit 0 file (if any), the
                 system enters the top-level loop. Each S-expression read

in is evaluated and the result is displayed. Note that
the top-level evaluator is EVAL, not EVALQUOTE.

The session can be terminated by calling the function
QUIT. If one wishes to terminate the session abnormally,
the function ABEND can be used.

| The complete description of UTILISP is given in MTS
Volume 22, UTILISP in MTS.

January 1987

*VALIDATEFILE

Contents: The line-file validation program.

Purpose: To check the internal consistency of MTS line files.

Use: The program is invoked by the $RUN command.

Program Key: *VALIDATEFIL

Logical I/O Units Referenced:
              GUSER - an input file name (if the PAR field is
                      omitted).
              SERCOM - messages to the user and the listing of defec-
                      tive lines if the file is inconsistent.

Parameter: A file name may be specified in the PAR field of the $RUN
              command.

Return Codes: 0 - File has been validated and appears consistent.
              4 - File has been validated and appears inconsistent.
              8 - File not validated (file does not exist or access
                  insufficient).

Description: The program accepts as input the names of line files, and
              checks their internal consistency. No changes are made
              to any file. If information for only one file is wanted,
              its name can be given in the PAR field of the $RUN
              command. If the parameter field is omitted, file names
              are read via GUSER until an end-of-file or null line is
              entered, or an attention interrupt is given.

              If the system crashes while a line file is being
              modified, there is a possibility that the file will be
              left in an inconsistent condition. A discussion of why
              these inconsistencies occur and how they can be recog-
              nized is given in Appendix E to "Files and Devices" in
              MTS Volume 1, The Michigan Terminal System. Continued
              use on an inconsistent file will only compound any
              existing problems.

              *VALIDATEFILE attempts to validate the internal consis-
              tency of a line file. Although it is not able to
              positively identify all problems, if it returns the
              message

                   FILE SEEMS OK

the probability is high that the file is undamaged. If
it finds the file to be inconsistent, it returns the
message

THIS FILE IS INCONSISTENT

The user may then enter the command LINES to obtain a
listing of the erroneous lines or EXPLAIN to obtain
information on how to go about repairing the damaged
file.

Users should not attempt to modify an inconsistent file.
Instead, the file may either be restored from a Computing
Center save tape or copied ($COPY) to another file.
Copying a file resolves the internal inconsistencies, but
erroneous lines of data must still be corrected in the
new copy of the file. *VALIDATEFILE will print the line
numbers of the lines in the file which are likely to be
erroneous.   This list is not necessarily exhaustive,
however. The file editor ($EDIT) can then be used to
edit the new copied file.

The cost of having an inconsistent file restored or
fixing an inconsistent file by copying and editing is
generally rebatable as is the cost of running *VALIDATE-
FILE to determine that the file is in fact inconsistent.
For each file, the cost of validation is printed by
*VALIDATEFILE.

Example: The example below illustrates the use of the context
                editor and *VALIDATEFILE to correct an inconsistent file
                which has the same line of information associated with
                two different line numbers. Input from the user is in
                lowercase; output from the system is in uppercase.

```
#$edit badfile
:print 3964
:   3964     A TYPICAL LYNE IN A F
:print 4042
:   4042     A TYPICAL LYNE IN A F
:alter 3964 'lyne'line'
:   3964     A TYPICAL LINE IN A F
:print 4042
:   4042     A TYPICAL LINE IN A F
:stop
#run *validatefile par=badfile
#EXECUTION BEGINS
 FILE:BADFILE
 THIS FILE IS INCONSISTENT
 TYPE "EXPLAIN", "LINES", OR "STOP"
 explain
```

January 1987

```
            TO PREVENT FURTHER DAMAGE, DO NOT WRITE IN THIS FILE.
            THE FILE MUST EITHER BE RESTORED FROM A COMPUTING CENTER
               SAVE-TAPE OR COPIED  TO  ANOTHER  FILE.   WITH  EITHER
            METHOD
               SOME  INFORMATION  MAY  BE  LOST.   TO  HAVE  THE FILE
            RESTORED
               CONTACT THE BUSINESS OFFICE AT THE COMPUTING CENTER.
            IF YOU WANT TO ATTEMPT TO FIX THE FILE YOURSELF,
               FIRST COPY IT TO ANOTHER FILE USING A COMMAND LIKE:
               $COPY BADFILE(FIRST)@-TRIM@-IC NEWFILE@INDEXD@-TRIM
               THEN EXAMINE NEWFILE FOR ERRORS.
            PLEASE CONTACT THE COUNSELORS IF YOU HAVE ANY
               QUESTION OR DOUBT.
            TO GET A LIST OF LINE NUMBERS THAT ARE LIKELY
               TO BE BAD, TYPE "LINES", OTHERWISE TYPE "STOP".
            lines
                 4008.000
                 4042.000
                 3964.000
                 4065.000
                 4202.000
                 4242.000
                 4082.000
                 $.28
            #EXECUTION TERMINATED
            #$create goodfile
            #  FILE "GOODFILE" HAS BEEN CREATED.
            #$copy badfile(first)@-trim@-ic goodfile@index@-trim
            #$edit goodfile
            :print 3964
            :  3964     A TYPICAL LINE IN A F
            :print 4042
            :  4042     A TYPICAL LINE IN A F
            :alter 3964 'line'lyne'
            :  3964     A TYPICAL LYNE IN A F
            :print 4042
            :  4042     A TYPICAL LINE IN A F
            :stop
            #$run *validatefile
            #EXECUTION BEGINS
             TYPE NAME OF FILE TO BE VALIDATED
             goodfile
             FILE:  GOODFILE
             FILE SEEMS OK
                 $.13
             TYPE NAME OF FILE TO BE VALIDATED
             $endfile
                 $.00
            #EXECUTION TERMINATED
```

January 1987

*VSS

Contents: The IBM OS/VS Operating System execution monitor.

Use: The program is invoked by the $RUN command.

Program Key: *VSS

Return Codes: See CCMemo 448.

Description: VSS is an execution monitor program that makes it
                 possible for MTS users to run application programs that
                 normally run under the IBM operating systems OS/MFT,
                 OS/MVT, VS1, and MVS. With the aid of simulation-
                 assisting features available under MTS, VSS accurately
                 recreates a fairly rich subset of the OS/VS environment,
                 thus allowing many ordinary OS/VS programs to run unmodi-
                 fied in MTS.

                 In order to provide an OS/VS environment without intro-
                 ducing foreign concepts into its enclosing MTS environ-
                 ment, VSS uses standard MTS system functions to do the
                 work of executing OS/VS system functions. Similarly, VSS
                 translates MTS system responses into equivalent OS/VS
                 system responses. In essence, VSS provides a two-way
                 mapping between the MTS way of doing things and the OS/VS
                 way of doing things.

                 VSS has a command language of its own that allows full
                 user control of the details of this mapping and provides
                 a number of convenience and debugging features.

                 VSS does not use OS/VS Job Control Language ("JCL"). The
                 VSS command language is similar to the command languages
                 of MTS and its subsystems. Consequently, VSS's language
                 is much simpler to use and easier to understand than
                 OS/VS Job Control Language.

                 The complete description of VSS is given in Computing
                 Center Memo 448, "VSS: OS/VS Simulator."

January 1987

*WATBOL

Contents: The University of Waterloo COBOL compiler.

Purpose: To compile and execute COBOL programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *WATBOL

Logical I/O Units Referenced:

        SCARDS - compiler control commands, COBOL source program, and program data (the default is *SOURCE*).

        SPRINT - compiler diagnostics, source program listing, and program output (the default is *SINK*).

        SERCOM - compiler error exit diagnostics (the default is *MSINK*).

        0   - execution-time library (the default is *WATBOL-LIB). This must be a library file produced by the *WATBOLLIBGEN program.

        SPUNCH - program output (optional).

        GUSER - program input data (optional).

        1-19   - program input data or output (optional).

        Note: Any logical I/O unit may be used for execution-time I/O by specifying its name in an ASSIGN clause, e.g.,

                ASSIGN TO UT-S-SPRINT

        In addition, SYSIN, SYSPRINT, and SYSOUT may be used in an ASSIGN clause: SYSIN will read data from SCARDS, and SYSPRINT and SYSOUT will write output to SPRINT.

Return Codes: Always zero.

Description: The WATBOL compiler was written, and is maintained, by the University of Waterloo, Waterloo, Ontario, Canada. It is a COBOL compiler designed for the beginning programmer: common programming errors, such as undefined variables, are flagged, and meaningful diagnostics are given. For a full description of the COBOL language, see IBM OS Full American National Standard COBOL, form GC28-6396.

Restrictions: The following features of ANS COBOL are not supported by WATBOL.

            (1) The REPLACING option of the COPY feature.
            (2) Only the TALLY, RETURN-CODE, LINE-COUNTER, and
                 PAGE-COUNTER special registers are available.
            (3) Sequence number checking.
            (4) RENAMES clause.
            (5) Object module punching.
            (6) I/O control statements: RERUN and SAME-AREA.
            (7) Declaratives (those not associated with the
                 Report Writer).
            (8) Tape label exit handlers.
            (9) Segmentation.
            (10) Debugging language.
            (11) Listing control features: SPACE1, SPACE2,
                 SPACE3.
            (12) Spanned records: RECORDING MODE S.
            (13) UNSTRING verb.
            (14) Teleprocessing feature.
            (15) SIGN clause.
            (16) Sterling currency feature.
            (17) ASCII support.
            (18) Device-dependent I/O support.
            (19) Special processing for multiple volumes or reels.

Compiler Options:

        The following compiler options may be specified in the
        PAR field of the $RUN command. These options may be
        given in any order and must be separated by commas. The
        maximum length of the PAR field is 72 characters.
        Abbreviations, where available, are underlined.

        BATCH
             The BATCH option indicates that each job in the
             input stream is preceded by a /COMPILE control
             command. Any input lines preceding the first /COM-
             PILE command are flushed by the compiler. If this
             option is not specified, the compiler will only
             compile and execute the first job even if it is not
             preceded by a /COMPILE command.

        BCD or NOBCD
             The BCD option specifies BCDIC input and the NOBCD
             option specifies EBCDIC input. The default is
             NOBCD.

        CHECK or NOCHECK
             The CHECK option causes undefined variable checking
             to be performed during program execution. The
             default is CHECK.

        CLINES=n
             "n" specifies the number of lines per page which

will be printed during the compilation listing. The default is 60.

KP={26|29}
    KP=26 is identical to the BCD option and KP=29 is identical to NOBCD. The default is KP=29.

LINES=n
    "n" specifies the maximum number of lines per page on output written to SPRINT or SYSPRINT.   The default is 60.

LIST or NOLIST
    The LIST option specifies that an object code listing of the assembled program is to be produced. The default is NOLIST.

MSGLEVEL=n
    The MSGLEVEL option controls compiler message print-ing. There are four levels of message types:

                    Level   Message Type

                      0   messages
                      1   warnings
                      2   extensions
                      3   errors

    MSG=0 will print all levels of messages; MSG=1 will print levels 1 through 3; MSG=2 will print levels 2 and 3; and MSG=3 will print only level 3. The default is MSG=1.

PAGES=p
    "p" specifies the maximum number pages to be printed during execution. The default is no page limit.

RUN or NORUN
    Ordinarily, if neither the RUN nor NORUN options are specified, the WATBOL job will execute only if there are no compile-time errors.   The RUN option will force program execution regardless of any compile-time errors detected. The NORUN option will prevent program execution altogether. Note that for some errors, program execution will always be suppressed even if RUN is specified, e.g., when object code overlaps the symbol table. The default is neither RUN nor NORUN.

SOURCE or NOSOURCE
    The SOURCE option specifies that a source program listing is to be produced. Note that source state-

ments containing errors will always be listed even
if NOSOURCE is specified. The default is SOURCE.

SUBCHECK or NOSUBCHECK
The SUBCHECK option causes subscript checking to be
performed during program execution. The default is
SUBCHECK.

TIME=s
"s" specifies the maximum time, in seconds, that a
given job in the batch stream can execute.   The
default is no time limit.

UCON=n
"n" specifies a nonnegative integer which is less
than 255. The hexadecimal equivalent of this number
will be used as the undefined value constant; that
is, the value all variables which have not had a
value assigned them by the WATBOL program will have.
Note that the default value, UCON=127, will print as
'"', and any other selected value should have this
characteristic.

D=(opt1,opt2,opt3,...)
The D option may be used to specify the default
values assumed for the options described above.
They will be used only if no explicit value is
specified in the input stream. These options must
not contradict values specified by the M option
described below, e.g., M=(T=50),D=(T=100) would be
in error.   Values specified in the WATBOL input
stream will override values set by the D option
provided they do not exceed the M values. Any
options specified by M, but not specified by D, take
both their default and maximum values from M.

M=(opt1,opt2,opt3,...)
The M option may be used to specify the maximum
values permitted for the options described above.
For numerical value options, e.g., TIME, this field
specifies the maximum value that the option can have
during the run. Values specified for those options
in the WATBOL input stream cannot exceed these
maximums or WATBOL will flag them as errors. For
nonnumerical options, e.g., CHECK, this field speci-
fies the forced state an option will have during the
run. These states cannot be overridden by an option
specified in the input stream.

Control Commands:

The following WATBOL control commands may be specified in
the input stream.   These commands are interpreted by

WATBOL and not by MTS. Each command must start in column one.

/COMPILE option-list

    The /COMPILE command initiates program compilation. The option-list is a series of compiler options that may be selected from the list of compiler options described above, but cannot include BATCH, D, or M. The /COMPILE command is not required if the BATCH compiler option has not been selected.

/EXECUTE

    The /EXECUTE command initiates program execution. This command must appear before the program data (if any). This control command is not required if there are no data.

/DATA

    The /DATA command is a synonym for the /EXECUTE command.

/STOP

    The /STOP command terminates the compiler. This is the last command in the input stream.

/ATTACH

    The /ATTACH command allows arbitrary MTS files or devices to be attached to the COBOL file. The assignment is denoted by

        COBOLFILE=MTSFDname.

    COBOLFILE may then subsequently appear in a SELECT clause in a COBOL program, and the COBOL file SELECTed will be attached to the MTS file; e.g., if the command

        /ATTACH MASTER=1EHB:PROBLEM1DATA

    has been specified,

        SELECT TRANSACTION
            ASSIGN TO UT-S-MASTER.

    will connect the COBOL file TRANSACTION with the MTS file 1EHB:PROBLEM1DATA

/OPTIONS

A list of options should appear, following the "/OPTION" text, selected from the compiler options described above. Any valid option may appear except for BATCH, D, LINES, M, RUN, TIME, and PAGES. These options may be used to change the state of compilation parameters during the course of a compilation. The locations in the job where options may appear are:

(1) between level numbers in a structure, e.g.,

```
      01  X  PICTURE IS 999.
/OPTIONS NOCHECK,LIST
      03  Y  PICTURE IS 99.
```

(2) between any two procedure division statements, i.e., the option must not occur in the middle of a statement.

(3) before any statement which validly starts in margin A or before any procedure division verb.

Examples:
```
$RUN *WATBOL
/COMPILE PAGES=4
      (source program statements)
/EXECUTE
      (program data)
/STOP
```

The above example illustrates the compilation and execution of a single WATBOL program.

```
$RUN *WATBOL PAR=BATCH,D=(CLINES=50)
/COMPILE
      (source program # 1 statements)
/DATA
      (data for program # 1)
/COMPILE
      (source program # 2 statements)
/EXECUTE
/STOP
```

The above example illustrates the batching of two WATBOL programs. Each program is compiled and executed as a separate and distinct program. The default number of compilation lines per page is 50.

January 1987

*WATBOLLIBGEN

Contents: The WATBOL library-generation program.

Purpose: To generate libraries used by the WATBOL compiler.

Use: The program is invoked by the $RUN command.

Program Key: *WATBOLLIBGE

Logical I/O Units Referenced:
                SCARDS - the description of the library to be produced.
                SPRINT - verification output.
                SPUNCH - the generated WATBOL library.
                SERCOM - terminal error messages and warnings.

Parameters: The following parameters may optionally appear in the PAR
                field, separated by commas or blanks, in the order
                presented below.

                NOVERIFY (or any proper initial substring thereof) - this
                suppresses the verification output on SPRINT.

                Any definition record, defined above, minus the "/"
                character. If present, the input on SPRINT is assumed
                free of all definition records and is placed into the
                library file as defined by the PAR field definition
                record. This parameter is provided for simpler usage
                when only one element is to be placed into the library
                file, which eliminates having a definition record in the
                SCARDS input text.

Return Codes: Always zero.

Description: The Version 1 Level 2 release of WATBOL, the University
                of Waterloo (Waterloo, Ontario, Canada) COBOL compiler,
                provides four sources for presenting input to the compil-
                er. The most commonly used is the logical I/O unit
                SCARDS, from which the compiler reads source statements
                comprising the program, and possibly program data. A
                WATBOL library file provides the three other sources,
                namely:

                    (1) COBOL source code incorporated into the compila-
                         tion of the COBOL program via the use of the
                         COBOL COPY verb,
                    (2) non-WATBOL subroutines, compiled (or assembled)
                         in some other language and made available to the
                         COBOL program as object modules for external
                         subroutines,

(3) list sections, text of a purely informative
     nature listed at the end of the compiled program
     text, but not compiled with the program.   List
     sections are available as a facility within
     WATBOL for the distribution of information per-
     tinent to WATBOL users.

Section Types

Each of these types of sources of input are contained in
a section of a WATBOL library file, which must be an MTS
file of the line or sequential type. A library file can
contain an (almost) arbitrary number of these sections.
The three section types are COPY sections, object module
sections, and list sections.

COPY Sections:

     COPY sections are collections of COBOL source state-
     ments included into the COBOL program through the
     use of the COPY verb in the COBOL language. These
     are read in from the WATBOL library as they are
     encountered in the program WATBOL is compiling.
     WATBOL expects to find a COPY section in the library
     with the name given after the COPY verb in the COBOL
     program; hence, each COPY section must have a name.
     COPY section text must be of no greater length than
     80 characters per record, and must conform to the
     standard   COBOL   source   statement   formatting
     conventions.

Object Module Sections:

     After WATBOL compiles the COBOL source program, it
     may have a collection of subroutine names which have
     been referenced by the program, but for which it has
     not encountered any COBOL code. If so, the subrou-
     tines are assumed to be object module sections
     within a WATBOL library.

     The text WATBOL reads from a library object module
     section must be an object module generated by some
     assembler or compiler. The loader records compris-
     ing the object module can only be ESD, TXT, RLD, and
     END records. Furthermore, they must be no greater
     than 80 characters in length, and cannot reference
     any external subroutines in MTS.   (The linkage
     editor can be used to change the record lengths in
     any object modules containing records longer than 80
     characters.) Subroutine linkage conventions are an
     extension of the standard FORTRAN conventions; FOR-

TRAN subroutines are callable from WATBOL-compiled
COBOL programs.

List Sections:

List sections are sections of text listed after the
compilation of a COBOL program. They are accessed
by WATBOL depending on the program's PROGRAM-ID. If
the PROGRAM-ID begins with the string 'COB', WATBOL
searches the library for a list section with the
PROGRAM-ID as its name. If one does not exist, it
is not considered an error. If found, the text
constituting it is listed after the program source
code. The text is a maximum of 80 characters wide.
Column 1 of the text is considered a carriage-
control column, with blank, '1', '0', and '-' being
valid there. Column 72 is a continuation column.
If nonblank, the text in the line following a line
with a nonblank column 72 is printed immediately
following column 71 of the previous line. Only one
continued line is allowed in succession.

The Library-Generation Program

Supplied with a name of a COPY section, list section, or
external subroutine which it wants to find, WATBOL reads
in a directory contained at the beginning of the library
file which contains the names of the sections and
subroutines and their storage location in the library
file. WATBOL searches the directory to find and read
that section or subroutine. The directory must be in one
of two formats, the format depending upon the type of the
MTS file, line or sequential.

The WATBOL library-generation program *WATBOLLIBGEN is
provided to build this directory and place it and the
text of the different section types into a file, thereby
creating a WATBOL library. *WATBOLLIBGEN takes a de-
scription of the library as input from the logical I/O
unit SCARDS and produces an actual library as output on
the SPUNCH logical I/O unit. A description of a library
is a series of records describing each section to be
placed into the library file. Each section is one of the
three types described above, and is input as follows:

    (1) 1 record, the definition record, giving the type
         and name, if necessary, of the section, followed
         by
    (2) n records of the actual text to be filed in the
         library section.

The definition record is one of the following:

        COPY SECTION name
        LIST SECTION name
        OBJECT MODULE
        OBJECT MODULES

which defines the characteristics of the section to be
placed in the library.   Column 1 of each definition
record must contain the identifying character '/' to
distinguish it from the text of the section preceding it.
The name on the COPY SECTION record is the name by which
one references the COPY section text with the COBOL COPY
verb.   On the LIST SECTION record, it is the PROGRAM-ID
which will trigger the listing of the list section text
after the program source. Object module sections cannot
be explicitly named because they are named by the name(s)
of the subroutine(s) they contain.

Besides producing a WATBOL library as output, *WATBOLLIB-
GEN also produces (optionally) a listing of the names of
all the sections cataloged into the library, plus the CPU
time needed to do so, on SPRINT.

Any error messages are printed on SERCOM.

Running WATBOL with a WATBOL Library File

WATBOL libraries are attached to logical I/O unit 0,
which is assigned to a library file on the MTS $RUN
command; e.g., if a library file to be used by WATBOL
were named LIBRARY, it would be specified by

    $RUN *WATBOL 0=LIBRARY [plus other I/O assignments]

WATBOL has a public library *WATBOLLIB in which it keeps
some subroutines it occasionally needs for the execution
of a COBOL program, a list section COBBUGS which contains
a list of currently known compiler bugs, and a list
section COBRWORD which contains a list of reserved words
not permitted as WATBOL identifiers.   *WATBOLLIB never
needs to be specified on the $RUN command -- it is always
available to WATBOL.   Any private library specified on
I/O unit 0 is searched before *WATBOLLIB for any request-
ed library sections, which permits private redefinition
of any sections in the public library. *WATBOLLIB is
only searched if the private library did not contain the
needed section.

WATBOL Library Internal Formats

A WATBOL library file consists of one or more directory
records at the head of the file, followed by the text of
the library sections. OBJECT MODULE text is placed into
the library file wholesale. COPY SECTION text is also
placed into the file as is, but a one record end-of-
section terminator, '/EOF', is inserted after the last
line of the section. LIST SECTION text is written into
the library file with a LIST-section identifying record,
'/LIST', placed in front of the list-section text, plus
the '/EOF' end-of-section terminator after the text.
This format does not differ from line to sequential
files.

The directory structure does differ, however. Line file
directories are identical to those produced for the
Assembler G: one record per library section, each
containing an 8-character, left-justified section title
or subroutine name, followed by 1 blank, followed by an
optionally signed 5-digit <u>integral</u> line number in the
file where the beginning of the section's /subroutine
text can be found.

Sequential file directories consist of one record. This
record contains one 12-byte block for each section, each
block containing 8 bytes of the section/subroutine name,
left-justified, followed by 4 bytes of POINT information
indicating the first line of the section's text in the
library file. Since a maximum line length of 32767 bytes
is possible for a sequential file, this puts a limit of
int(32767/12) = 2730 elements in a sequential library
file.

WATBOL External Subroutine Linkage Information

WATBOL calls external subroutines using a modified FOR-
TRAN calling convention. That is, at the time of the
call, general register 13 points to an 18-word save area,
general register 14 contains the address to which the
called subroutine should return, and general register 15
contains the address of the entry point of the
subroutine.

In a WATBOL call (as in a FORTRAN call), general register
1 points to a parameter list. Whereas FORTRAN parameter
lists contain just the addresses of the passed parame-
ters, the parameter list that WATBOL passes to an
external subroutine contains the address of the parame-
ters, plus the length of the parameters, in the following
format:

```
                    fullword address of parameter 1,
                    fullword address of parameter 2,
                         .
                         .
                    fullword address of parameter n, with the high-order
                        bit set to 1,
                    halfword length of parameter 1,
                    halfword length of parameter 2,
                         .
                         .
                    halfword length of parameter n.
```

The lengths in the parameter list are used internally by
WATBOL; they represent the number of bytes in memory that
the corresponding parameter occupies. If no parameters
are passed by the calling routine, general register 1
contains a zero.

WATBOL expects all the general registers to be restored
to their contents at the time of the subroutine call,
except general register 15, which may contain a halfword
return code which becomes the contents of the RETURN-CODE
special register. This means that the subroutine cannot
return a value in general register 0.

Examples: $RUN *WATBOLLIBGEN SPUNCH=library-file
               /COPY SECTION DEMO1
                    .
                copy section text
                    .
               /LIST SECTION COBBUGS
                    .
                list section text
                    .
               /OBJECT MODULES
                    .
                one or more external subroutine object modules
                    .
               /COPY SECTION DEMO2
                    .
                more copy section text
                    .
               $ENDFILE

               This will place DEMO1, COBBUGS, DEMO2, and any
               subroutine names in the object modules into the
               library file directory and the text of those sec-
               tions in the library file.

January 1987

                              *WATFIV

Contents: The University of Waterloo FORTRAN IV compiler.

Purpose: To compile and execute FORTRAN IV programs.

Use: The compiler is invoked by the $RUN command.

Program Key: *WATFIV

Logical I/O Units Referenced:
                SCARDS - compiler input (control commands and source
                         program) and data (defaults to *SOURCE*).
                SPRINT - compiler output (source listing and diagnostic
                         messages), program output, and job accounting
                         information (defaults to *SINK*).
                SPUNCH - program punch output.
                SERCOM - error comments and prompting messages in conver-
                         sational mode (defaults to *MSINK*).
                0   - library of subroutines (defaults to *WATLIB).
                1-19   - program input and output (default for input is
                         SCARDS assignment and default for output is
                         SPRINT assignment).

Parameter: The following parameter may be specified in the PAR field
                of the $RUN command.

                SIZE=n   The number of pages of virtual memory to be
                         used for the program(s). The default is
                         SIZE=10.

Return Codes: 0 - Successful return.
                3 - Syntax errors.
                4 - Execution errors.

Commands: The following are the most commonly used control commands
                for WATFIV. See the section "WATFIV" in MTS Volume 6 for
                a complete description of all of the control commands.

                /COMPILE initiates program compilation. This command
                          must appear before the program source state-
                          ments. Compiler options may be included on
                          this command (see below).

                /DATA   initiates program execution. This command must
                /EXECUTE appear after the last WATFIV statement and
                          before the data records (if any). Lines fol-
                          lowing this command are treated as data for the
                          current job. The end of the data is signalled
                          by the next control command or by an end-of-

                        file.   This command must be present even if
                        there are no data records.

            /STOP   terminates the compiler. An end-of-file also
                        terminates the program.

            /MTS   returns control to MTS. A $RESTART command may
                        be used to resume the program.

Options: The following compiler options may be specified on the
            /COMPILE control command. These options must be sepa-
            rated from the /COMPILE by at least one blank and from
            each other by commas or blanks.   Abbreviations, where
            permissible, are underlined.

            LIST   If specified, a listing of the source program
            NOLIST   is printed on SPRINT; if NOLIST is specified,
                        the listing is suppressed. The default is LIST
                        in batch mode and NOLIST in conversational
                        mode.   SOURCE is a synonym for LIST; NOSOURCE
                        is a synonym for NOLIST.

            LIBLIST   If specified, subroutines retrieved from the
            NOLIBLIST   library on unit 0 will be listed on SPRINT.
                        The default is NOLIBLIST.

            WARN   If specified, all extension messages will be
            NOWARN   printed on SPRINT; if NOWARN is specified,
                        warnings will be suppressed.   The default is
                        WARN.

            EXT   If specified, all extension messages will be
            NOEXT   printed on SPRINT; if NOEXT is specified,
                        extension messages will be suppressed.   The
                        default is EXT.

            TIME=s   The time estimate in seconds for the execution
                        of the WATFIV compiled program, where "s" is an
                        integer or decimal number. The default is the
                        time remaining in the MTS time limit (global or
                        local) minus one-fifth second.

            PAGES=n   The printed page estimate for output of the
                        WATFIV program. The default is the number of
                        pages remaining in the MTS page limit (global
                        or local) minus one page.

            LINES=n   The number of lines printed per page.   The
                        default is 60.

            RUN=FREE   This controls the amount of error checking done
            CHECK   by WATFIV.
            NOCHECK

January 1987

Description: See the section "WATFIV" in MTS Volume 6, <u>FORTRAN in MTS</u>,
                for a complete description of the WATFIV compiler.

                The file *WATLIB contains a library of WATFIV-coded
                functions and subroutines. This file should be assigned
                on the $RUN command to logical I/O unit 0, e.g.,

                        $RUN *WATFIV 0=*WATLIB

January 1987

<u>*WBASIC</u>

Contents: The University of Waterloo BASIC System.

                 Also see *BASIC in this volume for the description of the
                 University of Michigan BASIC System.

Use: The Waterloo BASIC system is invoked by the $RUN command.

Program Key: *WBASIC

Logical I/O Units Referenced:

                 None by the WBASIC system; however, any of the MTS
                 logical I/O units may be assigned by the user on the $RUN
                 command and referenced by the user while using WBASIC.

Parameters: See the parameter field description in the following
                 section on WBASIC-MTS system dependencies.

Return Codes: 0 - Successful return.
                 >0 - Size request too large or internal WBASIC error.

Description: This program is intended for use as a self-contained
                 system for debugging, modifying, and running programs
                 written in the BASIC language. Terminal input to WBASIC
                 is read from the MTS pseudodevice *SOURCE*.   Terminal
                 output from WBASIC is written to the MTS pseudodevice
                 SINK.

                 There are differences between Waterloo BASIC and the
                 University of Michigan BASIC (*BASIC). They are sum-
                 marized later in this description.

                 For more information, see <u>Waterloo BASIC - A Structured
                 Programming Approach, Primer and Reference Manual</u>, J. W.
                 Graham, et al, WATFAC Publications Ltd.,   Waterloo,
                 Ontario, Canada, 1980, IBSN 0-919884-22-9 which is avail-
                 able in the bookstores.

                 Note that chapters P and Q of the Waterloo BASIC Manual,
                 which deal with system dependencies, are invalid for
                 using *WBASIC in MTS. The following pages of this
                 description are replacements for those chapters.

WBASIC-MTS System Dependencies - IBM 370/303x/43xx

Overview

This supplement is intended to <u>replace</u> chapters P and Q in the Waterloo BASIC Primer and Reference Manual. It describes the features of Waterloo BASIC that apply only to the version implemented on the MTS operating system. Broadly speaking, these system dependencies apply to the file system and to the sign-on and sign-off procedures.

Signing On and Off BASIC

Once signed on to the MTS system, the user can invoke Waterloo BASIC by typing the following command:

        $RUN *WBASIC

To return to the MTS level, the user may issue either the BYE or FINISH command.

When WBASIC is invoked, it automatically searches for a special execute file (belonging to the user) with the name "WBSC.PROFILE". If the file exists, WBASIC processes commands and/or statememts from that file; otherwise, processing continues.

When WBASIC is active, programs may be brought into the workspace, manipulated, and run using commands provided for those purposes. Certain operations may be specified in the PAR field of the $RUN command:

        $RUN *WBASIC PAR=options

These options are described below.

'filename'   'filename', if specified, refers to either a source-
                program file (OLD) or an execute file (EXEC) contained in
                an MTS file by that name, e.g., 'PROG1' or 'EXFILE2'.

OLD/EXEC   OLD specifies that 'filename' contains a source program
                to be compiled into the workspace. EXEC specifies that
                'filename' contains commands and/or statements to be
                processed. EXEC must be specified if 'filename' is an
                execute file. The default is OLD.

RUN/NORUN   RUN specifies that the source program loaded from
                'filename' is to be automatically run. NORUN specifies
                that the program is not to be automatically run. The
                default is RUN.

BYE/NOBYE   BYE specifies that WBASIC is to be terminated (unloaded)
                after the program in the specified OLD file has been

January 1987

run.NOBYE specifies that control is to be retained by
WBASIC after the program is run.

VERBOSE/TERSE   VERBOSE specifies that all informational messages are to
be displayed. TERSE suppresses the display of the
log-on, 'Execution begins', and 'Execution terminated'
messages.

MTS/NOMTS   MTS allows MTS commands to be executed by using the MTS
built-in function in a BASIC program or by issuing them
at a terminal by prefixing the commands with a "$".
NOMTS does not allow the execution of MTS commands (e.g.,
disallows $PERMIT).

Workspace Size (The SPACE=nP Parameter)

   By default, WBASIC provides a workspace that will run a BASIC program
of approximately 150 lines (depending on sizes of matrices, complexity
of lines, etc.). To increase the size of the workspace, use the SPACE
parameter in the PAR field of the $RUN command. This parameter can be
separated by a comma or spaces from any other items in the PAR field.
The default is 8 pages and the maximum size is 256 pages. For example,
to double the workspace size use the following $RUN command:

      $RUN *WBASIC PAR=SPACE=16P

Files

   Waterloo BASIC uses the MTS file system for storage of either line or
sequential files for WBASIC's use. Any legal MTS file or device name
may be used in an OPEN statement. In addition, logical I/O units set on
the $RUN command may be referenced by enclosing the logical I/O unit
name or number in parenthesis in the open statement. For example, to
read lines from the file or device attached to logical I/O unit SCARDS,
use the following WBASIC open statement:

      OPEN #2,'(SCARDS)',INPUT

   WBASIC will issue a mode error message if either of the following is
attempted:

   (1) Opening anything but a line file for INOUT. WBASIC must be able
        to do indexed I/O to an INOUT file.

   (2) Opening an output-only device (e.g., *PRINT* for INPUT or
        INOUT).

   A file can be created from WBASIC by using the MTS command $CREATE or
by opening the file for OUTPUT. The Waterloo BASIC Manual in Section
H-5 on File Positioning states that if records are to be accessed
randomly using the REC clause, they should have been previously written

on the file sequentially. This is not necessary in Waterloo BASIC under
MTS.   An indexed read will always return an End-of-File if the record
does not exist. Note that this is similar, but not identical, to the
operation of Waterloo BASIC under CMS described in Section H-5.

   The specification of logical record lengths, record formats, and
block sizes as prescribed for the OPEN statement description in Section
H-5 and other sections of the Waterloo BASIC manual should be disre-
garded by the *WBASIC user in MTS. References to parameters such as
RECSIZE, LRECL, and RECFM are system-dependent (i.e., for the CPS System
and VM/CMS System).   OPEN statements in MTS should refer purely to an
MTS file/device name (e.g., 'FDATA1', '-TEMP', or '*PRINT*') or to an
MTS logical I/O unit name or number (e.g., '(SCARDS)' or '(12)').

   The number in the REC clause of a PRINT or INPUT statement is rounded
to three places of decimal before being used as an MTS line number. For
example:

     PRINT #3,REC=PI,PI

would write the value of PI in line 3.142. The RECNUM built-in function
may not be used without a _file-ref_ number as indicated in Section H-16.

   The Waterloo BASIC Manual assumes that sequential reads and writes
start at line zero. Under MTS they start at line one.

Keyed Files

   A keyed file is create by an OPEN statement in which the names of
both the index and data file must be specified. The data-file name is
then saved in the index file so that the name of the data file is
subsequently not needed when using the keyed file for INPUT or INOUT.
If the keyed file is to be used from another ID, this user's ID should
be included in the name of the datafile. For example, if this user's ID
is THOR,

     OPEN #7,'KLASS',OUTPUT,DATAFILE='THOR:KLASSDAT',KEYLEN=4

would create KLASS as the index file and KLASSDAT as the data file with
the name THOR:KLASSDAT being entered in the index file KLASS. If user
ZEUS wishes to read the data file KLASSDAT then THOR must $PERMIT both
KLASS and KLASSDAT READ to ZEUS. Then ZEUS can read KLASSDAT (assuming
use of unit 8) by opening the index file with

     OPEN #8,'THOR:KLASS',INPUT

and using an INPUT #8 statement with the KEY clause and other
appropriate list and, possibly, format information.

   WARNING: Under no circumstances should the user change the line
numbers in the data file (e.g., via $RENUMBER) since the index file's
access to the datafile would then be invalid.

January 1987

Sorting

   In MTS, Waterloo BASIC uses -SORTUTIL instead of SORUTIL as the
utility file. This file is used for all sorting, and the USING clause
of the SORT statement is ignored.

   When using TAGSORT with a nonkeyed file, the four-byte record number
at the beginning of each of the sorted records is the internal form of
the MTS line number. This is a one-word, binary integer representing
the MTS line number times 1000. For example, the MTS line number 2.000
would be represented as the binary integer 2000.

The WBASIC MTS Command and the MTS Built-In Function

   Certain MTS commands may be issued without leaving WBASIC. Any line
entered from the terminal, whether under program control or at command
level, that begins with a '$' is interpreted as an MTS command. This
facility is inhibited and will result in an error message if NOMTS has
been included in the PAR field of the $RUN *WBASIC command. The WBASIC
command "MTS" may be used to return to MTS without unloading WBASIC.
The $RESTART command returns control to WBASIC.

   MTS commands may be executed under control of a BASIC program using
the MTS built-in function. This function takes a single, string-valued
parameter which contains the command to be executed. The return value
of this function is always zero if the execution of MTS commands is
allowed; otherwise, the return code is 4. In the latter case, the user
has specified the NOMTS par field parameter. For example, to create a
10-page MTS line file named DATAF1 from a BASIC program at line 100, one
could have the following statement in the program:

     100 Perform_code = MTS('$CREATE DATAF1 SIZE=10P')

Screensize

   The internal system parameter SCREENSIZE determines the number of
lines to be displayed by the LIST, TYPE, and DIRECTORY commands before a
system pause is effected to allow the user to read the information on a
CRT-type terminal.   When the user signs on, the screen size is set to
zero (no pauses). This default was chosen to avoid conflict with the
paging mechanism provided by the 3278-type device-support routines in
MTS. Use the SCREENSIZE command to change this value.

   When WBASIC is pausing while processing the above commands, pressing
the return key will cause the listing to continue. Anything else will
interrupt the listing and will be interpreted as a command to WBASIC.

SLIMIT Command

   A facility has been provided for limiting the number of statements
that may be executed in each RUN of a program. This facility may be

used to prevent wasteful consumption of resources by runaway programs executing from unattended terminals. The command form is:

    SLIMIT <maxcount>

where <maxcount> is a positive integer constant to be used to limit the number of statements executed for a single RUN command. For example, SLIMIT 5000 limits program execution to five thousand statements. If the program exceeds this limit, it is interrupted and a diagnostic message is issued.

Mixed Case

   Programs and BASIC commands may be entered as either lowercase and/or uppercase characters. Lines are not translated to uppercase; therefore, variable names must be expressed consistently in whatever case the user chooses. For example, the variable Big_X is not the same as the variable Big_x.

Carriage Control

   Logical carriage control may be used for output to a file or device by appending the @CC I/O modifier to the FDname whenever it is specified for output. For example, the following two program segments are equivalent:

    10 OPEN #2, '*SINK*@CC',OUTPUT
    20 PRINT #2, '0A blank line precedes this line.'

    10 OPEN #2, '*SINK*',OUTPUT
    20 PRINT #2, ' '
    30 PRINT #2, 'A blank line precedes this line.'

See Appendix H to "Files and Devices" in MTS Volume 1, The Michigan Terminal System, for a complete description of logical carriage control.

Data Input to WBASIC

   Free-format data items can be separated only by commas. No blank separators are allowed. Fixed-format data items must be presented just as their format-image string specifies. Formatted output to a file may, depending on the context, produce necessary trailing blanks at the end of the record being written. For example, writing the number 12 as the last item of a record with the format '@###CR@' will produce 012¤¤ (¤ is a blank) as the last characters of that record. If the user copies that file (e.g. via $COPY) to another file without defeating the MTS TRIM option and then tries to read that record in the new file with the original format, WBASIC will flag the input operation as an error since the input record length is too short to satisfy the length requirement of the format item.

January 1987

DIRECTORY Command

   The DIRECTORY command gives a list of all the permanent files under
the user's ID. The command will also optionally take the same arguments
as the MTS $FILESTATUS command but they are to be enclosed, as a group,
in primes. This allows interrogation of the user's file status even
though the NOMTS parameter was specified in the PAR field of the $RUN
command.

        DIR   is equivalent to $FILESTATUS
        DIR 'DA?'   is equivalent to $FI DA?
                        (prints all file names beginning with DA)
        DIR 'DA?F?'   is equivalent to $FI DA?F?
        DIR 'DATAF1 FA' is equivalent to $FI DATAF1 FULLACCESS
                        (obtains all users' access to DATAF1)

Unsupported Features

   The following commands or statements are not supported in the MTS
version of Waterloo BASIC:

        ASSIGN   DEASSIGN   LOAD   LOCK
        SHOW   STORE   STOREOBJ   UNLOCK

Invocation Examples

        $RUN *WBASIC

        $RUN *WBASIC PAR=SPACE=30P

        $RUN *WBASIC SCARDS=DATAF1 5=-ODFILE PAR=NOMTS

        $RUN *WBASIC 5=INDFILE SPRINT=*PRINT* PAR=TERSE

        $RUN *WBASIC PAR='MYEXFILE',EXEC

        $RUN *WBASIC PAR='MYPROG',NOBYE

Features of Waterloo BASIC and U of M BASIC

The following summarizes the most important comparisons between Waterloo BASIC (*WBASIC) and U of M BASIC (*BASIC).

Waterloo BASIC

Waterloo BASIC is fully interpretive and uses BASIC statements to implement debugging. BASIC statements can be changed immediately when the program is being debugged. Breakpoints may be set by inserting PAUSE statements into the program. The input/output and assignment statements are used in immediate form to modify and display data. Since WBASIC is interpretive, directly uses MTS files for data, and provides an abundance of facilities, the program overhead cost of CPU time and memory is much higher than for U of M BASIC. However, WBASIC offers over U of M BASIC the advantages of large file data sets and programs; indexed, sequential, and keyed file access; structured programming constructs; a large set of built-in functions; long variable names; multi-dimensional arrays; multi-line, user-defined, internal functions with parameters and recursive capability; immediate (desk calculator) mode; CHAIN statement with parameters (no CALL statement); extended I/O status determination; and formatted I/O facilities.

U of M BASIC

U of M BASIC is a compile, load, and go facility that runs programs in machine code form and uses small, in-fast-memory data sets to effect a highly efficient execution. It also has a very comprehensive debugging facility which easily admits the setting and restoring of breakpoints, the stepping through or global tracing of program logic, and the simple modification and display of program variables. Variable references, however, are strictly in constant form, e.g., DISPLAY A(3) or MODIFY X 3. Program statements may be changed, but the program must be rerun to effect the changes. U of M BASIC provides advanced BASIC support for small-file I/O, matrix operations, and string operations, but not nearly to the extent that Waterloo BASIC does.

U of M BASIC was designed to provide a simple, MTS-like environment for newcomers who would be using the BASIC programming language as an introduction to computing and as a stepping stone to other languages and facilities of MTS. Hence, it has an elementary set of commands much like those of MTS, SDS, and the MTS Editor. It can be used for the efficient programming, running, and debugging of modest BASIC programs-

January 1987

*WIREWRAP

Contents: A universal computer-aided-design program for the docu-
                 mentation and construction of hardware logic devices.

Use: The program is invoked by the $RUN command. A subroutine
                 package containing a description of the hardware mounting
                 technique must also be loaded. This is done by either
                 specifying one subroutine package in a library file by
                 means of the PAR field, e.g.,

                     $RUN *WIREWRAP PAR=LIB=FDname,TYPE=name

                 or by concatenating a file containing a single subroutine
                 package to *WIREWRAP, e.g.,

                     $RUN *WIREWRAP+filename

Program Key: *WIREWRAP

Logical I/O Units Referenced:
                 SCARDS - source input.
                 SPRINT - output listings, printer plots, and error
                          comments.
                 SPUNCH - Gardner-Denver card output.
                 SERCOM - error comments if PAR=SERCOM is specified.

Parameters: The following parameters may be specified in the PAR
                 field of the $RUN command.   The parameters must be
                 separated by commas.

                 LIB=FDname   Specifies the library file to be searched
                              for loading the subroutine package (mounting
                              technique) specified in the TYPE parameter.
                              The default is *WWLIB.

                 TYPE=name   Specifies the subroutine package (mounting
                              technique) to be loaded. Defaults to the
                              first package in the library file (type LAB
                              in file *WWLIB).

                 SERCOM   Enables the printing of error comments on
                              SERCOM (defaults to *MSINK*). This is use-
                              ful if SPRINT output is directed to a file
                              or line printer.

Return Codes: Always zero.

Description: *WIREWRAP is a universal, general purpose computer-aided-
               design program that may be used to design devices using

         any manufacturer's logic series or wirewrap panel. It
         was written to assume tasks which range from the design
         of a logic device with a logic diagram to the actual
         implementation of that device in real hardware, and to
         produce and maintain documentation needed to service the
         device after it is built. Some of its features include:

              (1) Checking outputs of all gates and storage
                   elements for overloading.
              (2) Automatically deciding which modules (inte-
                   grated circuits) are required to implement the
                   device, listing them for the designer, and
                   calculating their total cost.
              (3) Deciding where these modules should be placed
                   on the wirewrap panel.
              (4) Establishing all wires that are required, mini-
                   mizing wire length wherever possible, generat-
                   ing wiring instructions which give end-points
                   and length of every wire or a card deck to
                   control an automatic wirewrapping machine.
              (5) Generating documentation including a dictionary
                   of signal-names and a diagram of the wirewrap
                   panel (bay).

         Besides these direct applications, *WIREWRAP can be used
         indirectly to derive cost comparisons of different imple-
         mentations of the same device, for instance, using a new
         MSI (Medium Scale Integration) Integrated Circuit to
         replace a group of Small Scale subassemblies into
         printed-circuit boards. Most importantly, it offers a
         design language which is completely independent of the
         hardware implementation used, so that the device-
         description cards need never be altered no matter how
         much the designer decides to revise his hardware or
         mounting techniques.

         Novice users should be aware that although there are
         significant advantages to using *WIREWRAP, there may be a
         "one-time" programming task necessary to create a sub-
         routine package and/or macro dictionary before the system
         may be used.   However, there are several public files
         available that may already contain packages desired by
         the user. The files associated with *WIREWRAP are:

              *WWICMAC
              *WWLABMAC
              *WWLIB
              *WWSTGRD1
              *WWSTGRD2
              *WWSQGRD

         The subroutine packages available in *WWLIB are:

January 1987

                        LAB   (ECE breadboarding suitcase)
                        RSERIES   (DEC wirewrap backpanel)
                        MSERIES   (DEC wirewrap backpanel)
                        IC   (DEC wirewrap backpanel)
                        MIC   (DEC wirewrap backpanel)
                        PG1   (Augat wirewrap PC board)
                        PG3   (Augat wirewrap PC board)
                        PG5   (Augat wirewrap PC board)
                        PG13   (Augat wirewrap PC board)
                        UG1   (Augat wirewrap PC board)

              For further details, see Computing Center Memo 267,
              "*WIREWRAP User's Guide."

Examples: $RUN *WIREWRAP SCARDS=DATA SPRINT=*PRINT* SPUNCH=*PUNCH*
              PAR=TYPE=MSERIES,SERCOM

              In the above example, source input is read from the
              file   DATA and output listings are written to
              PRINT. The Gardner-Denver output is punched on
              PUNCH.   MSERIES is used as the subroutine package
              for the mounting technique, and has been loaded from
              the library (*WWLIB) containing several subroutine
              packages.   Error comments will be produced on both
              SPRINT and SERCOM.

              $RUN *WIREWRAP+SUBRTNFILE SPRINT=OUTFILE

              In the above example, the source input is read from
              SOURCE (the default for SCARDS) and the output
              listings are written to the file OUTFILE.   SUBRTN-
              FILE contains the subroutine package for the mount-
              ing technique.

504   *WIREWRAP

January 1987

<u>*WORKSTATIONS</u>

Contents: A list of public terminals, microcomputers, and microcom-
               puter software available on the University of Michigan
               campus.

Use: The list may be obtained via the $COPY command, e.g.,

                   $COPY *WORKSTATIONS *PRINT*

January 1987

<u>*11ASR</u>

Contents: The assembler for Digital Equipment Corporation's PDP-11
                 computer.

Use: The assembler is invoked by the $RUN command.

Program Key: *11ASR

Logical I/O Units Referenced:
                 SCARDS - source input to assembler.
                 SPRINT - listing and error comments from the assembler.
                 SPUNCH - MTS loader records from assembler.
                 0   - test and timing records (optional).

Return Codes: Always zero.

Description: *11ASR translates source programs for the PDP-11 written
                 in the U of M PDP-11 Assembly Language. The object
                 modules produced are composed of MTS loader records which
                 must be processed by *LINK11 to produce an absolute tape
                 which can be loaded by DEC's absolute binary loader.
                 *11ASR has an extensive macro capability.

                 A description of the U of M PDP-11 Assembly Language, a
                 description of the assembler, and macro compiler is given
                 in Computing Center Memo 286.

Parameters: All parameters may be specified in the PAR field of the
                 $RUN command, separated by commas, or in the   PAR
                 pseudo-op.

                 The default parameters are:

                     Batch: NOESD,LIST,RLD,XREF,LONG,NOREP,WRN
                     Teletype: NOESD,NOLIST,NORLD,NOXREF,SHORT,NOREP,WRN
                     2741:   NOESD,NOLIST,NORLD,NOXREF,LONG,NOREP,WRN

                 If SPRINT is assigned to a file during a terminal
                 session, then the batch-mode defaults are used.   If
                 SPUNCH is not assigned, the NODECK option is automatical-
                 ly selected.   In any case, lines found in error are
                 printed on SPRINT along with the error comment.

                 The legal parameters are:

                   BATCH - allow the assembler to reinitialize itself
                            after processing an END statement in PASS2 so
                            that further assemblies may be processed.

              ESD   - print the external symbol table.
              NOESD   - suppress printing of the ESD table.

              LIST   - print the source statement listing.
              NOLIST - suppress printing of the source listing.

              RLD   - print the relocation dictionary.
              NORLD   - suppress   printing   of   the   relocation
                        dictionary.

              XREF   - generate and print a cross-reference table.
              NOXREF - suppress the generation and printing of the
                        cross-reference table.

              LONG   - print sequence numbers in the listing.
              SHORT   - suppress sequence numbers in the listing
                        (thus making the line shorter and the XREF
                        table useless).

              REP   - enable the replacement scanner.
              NOREP   - disable the replacement scanner.

              WRN   - print a warning message if a word instruction
                        refers to an odd address.
              NOWRN   - suppress the above warning.

              NODECK - suppress generation of the object module.

              TEST   - generate and write statement timing and flow
                        information on logical unit 0.

              ON   - turn on printing if PAR=LIST is specified.
              OFF   - turn off printing of the listing.

Reader's Comment Form

Public File Descriptions
Volume 2
January 1987
Reprinted August 1988

Errors noted in publication:

Suggestions for improvement:

Your comments will be much appreciated. Send the completed form to the
Computing Center by Campus Mail or U.S. Mail, or drop it in the
Suggestion Box at the Computing Center, NUBS, or UNYN.

Date ————————————————

Name ————————————————————————————————

Address ——————————————————————————————

————————————————————————————————

————————————————————————————————

Publications
Computing Center
University of Michigan
Ann Arbor, Michigan 48109

Update Request Form

Public File Descriptions
Volume 2
January 1987
Reprinted August 1988

Updates to this manual will be issued periodically as errors are noted
or as changes are made to MTS. If you would like to have these updates
mailed to you, please submit this form.

Updates are also available in the files at some of the Computing
Center's larger public stations such as NUBS and UNYN; there you may
obtain any updates to this volume that may have been issued before the
Computing Center receives your form. Please indicate below if you want
to have the Computing Center mail you any previously issued updates.

        Name ━━

        Address ━━

                ━━

                ━━

        Previous updates needed (if applicable):━━━━━━━━━━

Send the completed form to the Computing Center by Campus Mail or
U.S. Mail, or drop it in the Suggestion Box at the Computing Center,
NUBS, or UNYN. Local users should give a Campus Mail address.

                        Publications
                        Computing Center
                        The University of Michigan
                        Ann Arbor, Michigan 48109

Users associated with other MTS installations (except the University of
British Columbia) should return this form to their respective installa-
tions. Addresses are given on the reverse side.

Addresses of other MTS installations:

    Publications Clerk
    352 General Services Bldg.
    Computing Services
    The University of Alberta
    Edmonton, Alberta
    Canada T6G 2H1

    Information Officer, NUMAC
    Computing Laboratory
    The University of Newcastle upon Tyne
    Newcastle upon Tyne
    England NE1 7RU

    Rensselaer Polytechnic Institute
    Documentation Librarian
    310 Voorhees Computing Center
    Troy, New York 12181

    Simon Fraser University
    Computing Centre
    User Services Information Group
    Burnaby, British Columbia
    Canada V5A 1S6

    Wayne State University
    Computing Services Center
    Academic Services Documentation Librarian
    5925 Woodward Ave.
    Detroit, Michigan 48202