Chapter 10. Utilities

Summary of Utility Programs

This chapter describes various utility programs that are available to the user. First, the programs are listed by function, with a short description of each. Once you have decided which program you need, use the Index or Table of Contents to locate the detailed description of that utility.

Archiving

ARCHIV	Archives (dumps) a group of files to tape. Note: it is recommended that FILARC or EXARCH be used rather than ARCHIV.
EXARCH	Archives (dumps) a group of files to tape for exporting to a non-MUSIC installation.
EXREST	Restores one or more files from a dump created by EXARCH.
FILARC	Archives (dumps) a group of files to tape. This is the recommended utility to use for creating backup copies of files or for transporting files to other MUSIC installations.
FILCHK	Reads and checks a dump produced by FILARC. It can produce a list of file names contained in the dump.
FILRST	Restores one or more files from a dump created by FILARC.
RETREV	Restores one or more files from a dump created by ARCHIV.
UDSARC	Archives (dumps) a set of UDS files to tape.
UDSRST	Restores a UDS file from a dump created by UDSARC.

Working with Files

AMS	Supports the MUSIC/SP Virtual Storage Access Method (VSAM).
ENCRYPT	Encrypts (codes) files for added security.
DECRYPT	Restores encrypted files back to a readable form.
LBLIST	Displays the contents of all the user's files.
VIEW	Allows viewing on a full-screen workstation of files of any record length or type and of any size.
VMPRINT	Prints the contents of files on a specified printer.

Working with UDS Files

AMS	Supports the MUSIC/SP Virtual Storage Access Method (VSAM).
DSCHK	Reads and checks a dump produced by UDSARC. It can produce a list of the data set names contained in the dump.
DSCOPY	Copies from one UDS file to another.
DSLIST	Displays a list of the names of all the UDS files you own.
DSREN	Renames a UDS file.
UDSARC	Archives (dumps) a set of UDS files to tape.
UDSRST	Restores a UDS file from a dump created by UDSARC.

Sorting

MNSORT	Sorts the records of a specified file into ascending or descending order, using various parts (fields) of the records as keys. Other sorting facilities are described in the same section. These include subroutines DSORT, SSORT and SRTMUS, and sorting in Cobol and PL/I programs.

Batch Utilities

The following utility programs are described in *Chapter 3. Using Batch*.

OUTPUT	Manages print files in the MUSIC print queue.
PQ	Displays what is queued to print on the various printers.
PRINT	Prints a file on a designated printer.
SUBMIT	Submits a job to MUSIC batch or to other processors.

Miscellaneous Utilities

DEBUG	Enables you to debug programs running on MUSIC at the machine language level.
FTP	Transfers files with computers on the TCP/IP Internet.
PANEL	Provides easy access to full-screen I/O from high-level languages.
POLYSOLVE	Can be used as desk calculator and solves polynomial equations.
PROFILE	Changes or displays your sign-on userid attributes, passwords, and limits.
TAPUTIL	Dumps or summarizes selected files from a tape, and copies files from one tape to another.
TELNET	Establishes workstation sessions with computers on the TCP/IP Internet.
UTIL	Facility for listing, punching, copying, and merging data records. The records can be in sequential files on disk, tape, or cards. Record lengths up to 133 bytes can be processed.
ZERO.FILE	Provides an efficient way of writing binary zeroes to all blocks of a file or UDS file.

MUSIC/SP Access Method Services (AMS)

MUSIC/SP Access Method Services (AMS) is a utility program that supports the MUSIC/SP Virtual Storage Access Method (VSAM). It implements a subset of the command language described in the IBM publication *OV/VS Access Method Services*, GC26-3841.

Access Method Services is used to allocate, manipulate, and generally maintain VSAM datasets. VSAM datasets cannot be created using /FILE statements. As well, the editor and various file utility programs should not be used to update or modify VSAM files. These functions are provided by Access Method Services.

Since MUSIC/SP VSAM does not support VSAM catalogs, AMS does not support commands that manipulate the catalog. None of the modal commands are supported in MUSIC/SP AMS. A subset of the functional commands are supported; see below for a complete list.

MUSIC/SP AMS can be run interactively at the workstation, or submitted to MUSIC batch. AMS commands can be entered directly from the keyboard, or they can be stored in a file and executed.

Definitions and Basic Concepts

This section defines some common terms used in the processing of VSAM files. Following this, some basic concepts are explained. These terms and concepts should be understood prior to using Access Method Services to manipulate VSAM datasets. Access Method Services allocates VSAM clusters, which are more than simple datasets. A VSAM *cluster* is one of the following:

- a *key sequenced* file (KSDS) consists of a MUSIC dataset for the data component and another MUSIC dataset for the index component. These files are maintained by VSAM in logical key sequence and can be accessed directly via a logical key.
- an *entry sequenced* file (ESDS) consists of a single MUSIC dataset for the data component. Entry sequenced datasets can be regarded as "sequential" datasets. Records can be accessed by relative byte address (displacement from the beginning of the dataset).
- a *relative record* file (RRDS) consists of a single MUSIC dataset for the data component. These files allow you to store information into *slots*, and to access data by *slot number*. A *slot* is the number of a fixed length record relative to the beginning of the VSAM file.

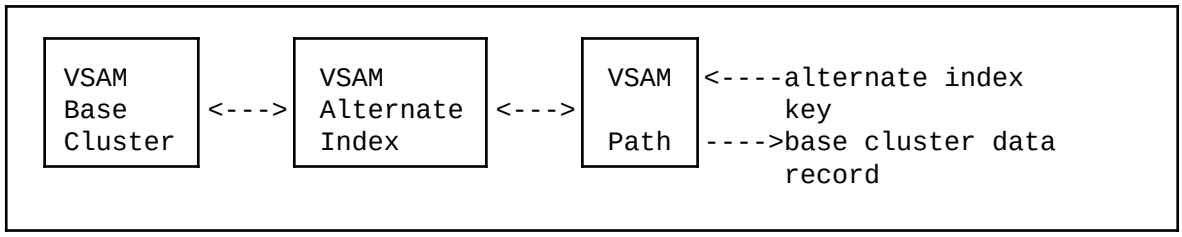
A key sequenced cluster has a field designated as the *primary key*. This key field must contain unique values; otherwise, VSAM will indicate an error condition. VSAM attempts to keep the data stored in the key sequenced cluster in the order of this primary key. Due to the patterns of additions and deletions, this may prove to be impossible. However, VSAM always presents the data in a key sequenced dataset in key-sequence order when processed sequentially.

An *alternate index cluster* is an auxiliary index built over some field in a VSAM *base cluster*. The base cluster and the alternate index cluster are logically related to each other by Access Method Services. VSAM automatically updates an alternate index when updates are made to the associated base cluster. Note that an alternate index has no meaning if the base cluster over which it is defined has been deleted.

Alternate indexes can be built over both KSDS files and ESDS files, but not over RRDS files. The cluster with which the alternate index is associated is termed the *base cluster*.

A *path* is a small MUSIC dataset that points to an alternate index to be used as if it were the primary key of a VSAM cluster. That is, when a path is opened, VSAM allows a processing program to make key sequenced

requests to the alternate index, and returns to the processing program records from the data component of the base cluster (refer to the diagram).



A *unique* key is a key that cannot have duplicate values. Usually, names of people do not result in unique key values, whereas Social Insurance Numbers do result in unique key values. VSAM key sequenced data sets (base clusters) **must** have unique keys; this is optional with alternate indexes.

The *upgrade set* of a base cluster simply consists of all alternate indexes that have been built over the base cluster, and have been flagged for upgrade processing (via the UPGRADE parameter which is the default). Whenever a base cluster is opened for output processing, each member of the upgrade set is opened as well.

To illustrate some of the above concepts, consider a program processing a VSAM path. It may require up to 5 MUSIC datasets (more if there are several members to the upgrade set). The following list enumerates them:

- A KSDS base cluster requires 2 datasets, one for the data component and one for the index component.
- The alternate index is a special type of KSDS and thus requires 2 datasets, one for the data component and one for the index component.
- A path requires 1 MUSIC dataset.

File sharing involves some special considerations. To allow any user to have READ access to a VSAM file, the SHARE attribute must be specified when the file is defined. To allow any user to have WRITE access to a VSAM file, the WRITE attribute must be specified when the file is defined. To actually OPEN a file for WRITE access by two or more users simultaneously, the WSHR option must be specified on the /FILE statement defining the file.

For more information on file sharing with VSAM files, refer to *Chapter 4. File System and I/O Interface* "MUSIC/SP Virtual Storage Access Method".

Command Language Syntax

Each Access Method Services command has the following general syntax:

```
command positional-parameter keyword1 keyword2 ... keywordn
```

Positional parameters may or may not be required. This is dependant upon the particular command. *Keyword* parameters are usually optional. However, certain information may be required by a particular command. As well, keywords which do not require a value do not have parentheses.

A command may be continued onto a subsequent line by using a continuation character. In Access Method Services, the hyphen ("-") is used as a continuation character. It must be placed after the text on a line, surrounded by blanks, and indicates that the command is to be continued on the next line. Note that values

may not be continued. Therefore, complete any name or other value on the same line that it starts. A maximum of 50 continuation lines is allowed.

Comments may be inserted anywhere within an Access Method Services command. Comments must be surrounded by the "/*" and "*/" sequence, similar to PL/1. Thus the string "/* comment */" is treated as a valid comment. Comments are treated as "white space" and may appear anywhere that a blank may appear. If a comment is not ended on a line, it is considered to continue onto the next line. Comments are continued until terminated by a "*/".

For example, the DELETE command requires a positional parameter, which is the dataset name to be deleted. If an alternate index is to be deleted, then this must be indicated via the AIX or ALTERNATEINDEX keyword. Since this keyword has no value (simply providing information to AMS), parentheses are not used, and the command is specified as shown. Note that the comment can be placed in the middle of the command.

```
DELETE   -
  filename  /* this is a comment */ -
  AIX
```

Invoking Access Method Services

Access Method Services is invoked by entering AMS when in *Go mode. On workstations that can accept 3270 data streams, Access Method Services displays a full-screen menu. On workstations that cannot support full-screen applications, a conversational interface is presented.

The full-screen interface has the ability to save the generated AMS commands in a file. By pressing a function key, these commands can be re-executed. An option is available to either append generated commands to this file, or to replace the current file each time a command is generated.

If for some reason you do not want to use the full-screen interface, you can suppress it by using the parameter NOFS when you invoke AMS. For example, the command AMS NOFS entered in *Go mode never invokes the full-screen interface to AMS.

You can directly execute a file of AMS commands by typing:

```
AMS filename
```

where **filename** is the name of the file containing your AMS commands. Doing this implies that the full-screen interface to AMS will not be invoked.

Note that AMS command stream files may also be submitted to MUSIC/BATCH.

Standard input is on logical unit 5, and standard output is on logical unit 6. Either unit number may be re-defined to refer to files as required, using /FILE statements.

In particular, to save the output of AMS to a file, use the following control lines:

```
/FILE 6 NAME(your-file) NEW(REPLACE) RECFM(VC)
/INCLUDE AMS
/INCLUDE your-AMS-commands-stream
```

Access Method Services Commands

The following Access Method Services commands are supported:

ALTER: modifies various attributes of a VSAM file.

BLDINDEX: constructs an alternate index for a base cluster.

DEFINE CLUSTER: defines a VSAM cluster.

DEFINE ALTERNATEINDEX: defines a VSAM alternate index.

DEFINE PATH: defines a VSAM path.

DELETE: deletes VSAM objects.

LISTCAT: displays the attributes of a VSAM cluster, and any associated files.

REPRO: copies VSAM or non-VSAM files to VSAM or non-VSAM files.

The following sections describe each command and the parameters that are supported. Many parameters described in the *OS/VS Access Method Services* are not supported. If these are specified, Access Method Services issues warning messages. Additionally, parameters specific to MUSIC/SP have been added.

ALTER

Usage

```
ALTER  entryname
       [FREESPACE(CI-percent)]
       [INDEXBUFFERS(number)]
       [INDEXPOINTER(newname)]
       [NEWNAME(newname)]
       [PRIVATE|SHARE|PUBLIC|COMMON]
       [RECORDSIZE(average maximum)]
       [RESETUPGRADELIST]
       [UNIQUEKEY|NONUNIQUEKEY]
       [UPGRADE|NOUPGRADE]
```

The ALTER command allows you to modify some of the attributes of a VSAM file. A single positional parameter specifying the name of the file to be altered is required. All other parameters are optional. Note that ALTER writes to the VSAM file, and thus changes the date that the file was last written.

The table in Figure 13.1 indicates all of the options available with ALTER, and the types of VSAM object that they can be used with.

Abbreviation: ALT

	ALTERNAME	INDEX	CLUSTER		PATH
	Data	Index	Data	Index	
COMMON	X	X	X	X	X
FREESPACE	X		X		
INDEXBUFFERS	X		X		
INDEXPOINTER			X		
NEWNAME	X	X	X	X	X
NONUNIQUEKEY	X				
NOUPGRADE	X				
PRIVATE	X	X	X	X	X
PUBLIC	X	X	X	X	X
RECORDSIZE	X		X		
RESETUPGRADELIST			X		
SHARE	X	X	X	X	X
UNIQUEKEY	X				
UPGRADE	X				

Figure 10.1 - Alter Keywords Used with VSAM Objects

The following describes each of the parameters.

entryname — specifies the name of the VSAM file to be altered.

COMMON — (COM) specifies that the *entryname* should be placed in the common index.

FREESPACE — (FSPC) specifies the percentage of a controlinterval's space that VSAM will maintain as freespace in future insertions. Note that this parameter does *not* cause every control interval in the data component to have the designated percentage of freespace.

INDEXBUFFERS — (IDXBUFS) specifies the number of index buffers that VSAM allocates when the VSAM dataset is opened. This parameter can affect performance. By maintaining more of the index in memory, more virtual storage may be required, but the probability that VSAM must access DASD to locate an index block may be lessened.

INDEXPOINTER — (IDXPTR) specifies that the *newname* is to replace the index pointer name contained in the first block of the data component. When you rename an index component, you must also update the index pointer field within the associated data component file. Note that the index component is required if the data contained within a

key-sequenced data set is to be retrieved.

NEWNAME (NEWNM) specifies that the VSAM object is to be renamed to *newname.* If the new name is invalid, then the rename operation is not performed. Associated VSAM objects are updated as required with the new name.

NONUNIQUEKEY (NUNQK) specifies that duplicate values should be allowed in the key field of an alternate index.

NOUPGRADE (NUPG) specifies that the alternate index should not be opened and updated to reflect changes made to the base cluster when it is opened for output processing. In other words, the alternate index is removed from the upgrade set of the base cluster.

PRIVATE (PRIV) specifies that the attribute flags for this file should be changed to PRIVATE (only the userid's owner can access the file).

PUBLIC (PUBL) specifies that the attribute flags for this file should be changed to PUBLIC (any user with any userid can access the file).

RECORDSIZE (RECSZ) specifies the maximum recordsize for this file. Note that the average recordsize value is ignored and may be specified as zero.

RESETUPGRADELIST
(RESETLST) specifies that the upgrade list within a VSAM cluster is to be emptied. This means that those objects are logically not to be associated with the base cluster, although they may still exist physically.

The intended use of this parameter is in the situation where VSAM files are being copied from one userid to another userid, perhaps by an archival and restore process. In this case, rather than change all pointers in all datasets (all of which may have different names due to the restore process), the recommended procedure is the following.

Delete all alternate indexes and paths, using the MUSIC/SP PURGE command. Then the list of members of the upgrade set contained in the data component of the base cluster should be emptied by using this parameter. Finally, the required VSAM objects previously deleted should be DEFINEd and any required indexes constructed using BLDINDEX. Refer to Example 2 for an illustration of this.

UNIQUEKEY (UNQK) specifies that duplicate values should not be allowed in the key field of the alternate index. Note that this parameter is not allowed if the alternate index is non-empty.

SHARE (SHR) specifies that the attribute flags for this file should be changed to SHARE (file can be accessed with the userid and name by anyone).

UPGRADE (UPG) specifies that the alternate index is to be placed in the upgrade set of the associated base cluster. Note that this parameter is not allowed if the alternate index is not empty. Thus, this parameter should be specified with an ALTER command prior to performing a BLDINDEX function upon it.

ALTER Examples

Example 1: Renaming a VSAM KSDS.

```
ALTER xxx NEWNAME(yyy)
```

Example 2: A VSAM KSDS and alternate index have been copied to another userid. The following steps will render the files usable from this userid (if these steps are not performed, VSAM OPENs will fail).

```
DELETE xxx.aix ALTERNATEINDEX
ALTER  xxx RESETUPGRADELIST
DEFINE AIX(NAME(xxx.aix)...)
BLDINDEX IDS(xxx) ODS(xxx.aix)
```

Example 3: Changing the maximum recordsize and freespace parameters.

```
ALTER xxx RECORDSIZE(4089) FREESPACE(5)
```

BLDINDEX

Usage

```
BLDINDEX  INDATASET(entryname)
          OUTDATASET(entryname)
```

The BLDINDEX command builds an Alternate index over a VSAM base cluster. Both the input file and the output file must be specified. The INDATASET parameter must name a valid VSAM base cluster, and the OUTDATASET parameter must name a valid VSAM alternate index.

BLDINDEX constructs an alternate index by reading all of the data records in the base cluster, then sorting them, and finally writing sorted alternate index data records to the alternate index.

The cost of running BLDINDEX can be minimized in several ways. First, since BLDINDEX performs a virtual storage sort if possible, specifying a larger REGION size on the /SYS statement can avoid access to DASD for temporary storage. Avoiding unnecessarily large key sizes will reduce overhead, since only the prime and alternate keys are used in constructing a temporary sort record.

BLDINDEX uses two work files for sorting the alternate key/primary key pairs. They are SORTWK1 and SORTWK2. You can over-ride the default files (SORTWK1 and SORTWK2) with job control statements.

Abbreviation: BIX

The following describes each of the parameters.

INDATASET (IDS) specifies the name of the VSAM base cluster over which the alternate index is to be built. This must be a KSDS or ESDS VSAM cluster. Note that this base cluster may not be a UDS file; alternate indexes over UDS files are not supported.

OUTDATASET (ODS) specifies the name of the VSAM alternate index which is to contain the alternate key information. Note that this file must previously have been defined using the DEFINE ALTERNATEINDEX command.

BLDINDEX Examples

Example 1: Building an alternate index using defaults

```
/INCLUDE AMS
BLDINDEX                                -
            INDATASET(BASEFILE.DAT)     -
            OUTDATASET(BASEFILE.AIX)
```

Example 2: Building an alternate index, using files, and specifying alternate work files

```
/FILE SORTWK1    UDS(XXXXWK1) OLD VOL(MUSIC1)
/FILE SORTWK2    UDS(XXXXWK2) OLD VOL(MUSIC1)
/INCLUDE AMS
BLDINDEX                                -
            IDS(BASEFILE.DAT)           -
            OUTDATASET(BASEFILE.AIX)
```

DEFINE ALTERNATEINDEX

Usage

```
DEFINE  ALTERNATEINDEX
          (NAME(entryname)
          RELATE(entryname)
          [CYLINDERS(primary[ secondary])|
             RECORDS(primary[ secondary])|
             TRACKS(primary[ secondary])]
          [CONTROLINTERVALSIZE(size|4096)]
          [FREESPACE(CI-percent[ CA-percent]|0 0)]
          [KEYS(length offset|64 0)]
          [PRIVATE|COMMON|SHARE|PUBLIC|WRITE]
          [RECORDSIZE(average maximum|4089 4089)]
          [REPLACE]
          [SPACE(primary|40)]
          [SECSPACE(secondary|0)]
          [UNIQUEKEY|NONUNIQUEKEY]
          [UPGRADE|NOUPGRADE])

        [DATA
          ([CONTROLINTERVALSIZE(size|4096)]
          [CYLINDERS(primary[ secondary])|
             RECORDS(primary[ secondary])|
             TRACKS(primary[ secondary])]
          [KEYS(length offset)]
          [NAME(entryname)]
          [RECORDSIZE(average maximum|4089 4089)]
          [SPACE(primary|40)]
          [SECSPACE(secondary|0)]
          [UNIQUEKEY|NONUNIQUEKEY])]


        [INDEX
          ([CONTROLINTERVALSIZE(size|512)]
          [CYLINDERS(primary[ secondary])|
             RECORDS(primary[ secondary])|
             TRACKS(primary[ secondary])]
          [NAME(entryname)]
          [SPACE(primary|10)]
          [SECSPACE(secondary|0)])]
```

The DEFINE ALTERNATEINDEX command creates a VSAM alternate index associated with an already existing base cluster. This consists of a data component and an index component.

Keywords that occur in both the ALTERNATEINDEX clause and the DATA clause apply only to the DATA clause. Those keywords need not be specified in both clauses. If a keyword appears in both clauses, the keyword in the DATA clause is used. The reason for this is that if AMS command streams from other systems are used, the specification intended for the data component will be used for the data component. Abbreviation: DEF AIX

The only required parameters are the NAME and the RELATE subparameters in the CLUSTER keyword. The following describes each of the parameters.

COMMON (COM) specifies that the file names for the alternate index are to be placed in the common index.

CONTROLINTERVALSIZE
(CISIZE) specifies the controlinterval size to be used for the DATA and/or INDEX component of the VSAM alternate index (default: 4096 for data component, 512 for index component).

CYLINDER (CYL) specifies the allocation of space for the data component in terms of cylinders of a 3330 disk pack (this is 19 tracks of 20 512-byte blocks each).

FREESPACE (FSPC) specifies the freespace percentages for control intervals. Note that although the control area freespace percentage is scanned, it is ignored.

KEYS specifies the length and offset of the key of the VSAM alternate index. Note that the offset is relative to 0, so that if the key starts in column 1, the value that you provide should be 0.

NAME (NA) specifies the name of the VSAM alternate index. When used in the ALTERNATEINDEX keyword, it specifies both the name of the VSAM data component and the default root for the index component of the alternate index. Note that the data component name is the one used when opening the file or when using the file name on a /FILE statement. In the special case of a name specified in the ALTERNATEINDEX keyword as "name.DAT", Access Method Services provides a default name for the index component of "name.IDX". Otherwise, Access Method Services appends ".IDX" to the name of the data component by default.

NONUNIQUEKEY (NUNQK) specifies that the alternate index key can have repeating values in the base cluster data records (this is the default).

NOUPGRADE (NUPG) specifies that the alternate index being defined should not be placed in the upgrade set for the associated base cluster.

PRIVATE (PRIV) specifies that the VSAM objects are to be given a file attribute of PRIVATE, and therefore cannot be shared with other users (this is the default).

PUBLIC (PUBL) specifies that the VSAM objects are to be given the file attributes of COM and SHR, meaning that non-owners can read from the file, and that the owner's userid need not be prefixed to the file name.

RECORDS (REC) specifies the space allocated to the data component in terms of the maximum logical recordsize (default: CISIZE - 7) and a secondary allocation in terms of the maximum logical record size.

RECORDSIZE (RECSZ) specifies the average logical recordsize and the maximum logical record-size for the data component. Note that the average logical recordsize is scanned but ignored.

RELATE (REL) specifies the base cluster over which the alternate index is being defined (note that this file must exist).

REPLACE (REPL) specifies that if the VSAM objects being defined already exist, then they are replaced (note that this is NOT the default; in other words, if the VSAM objects

being defined exist, the DEFINE command terminates with an error condition).

SECSPACE — (SECSP) specifies the MUSIC secondary space allocation in units of 1024 bytes (K) for the data or index component. If 0 is specified (the default), this means that a secondary space allocation will be 50% of the current file size.

SHARE — (SHR) specifies that the VSAM objects being created are to be given the SHR attribute, meaning that non-owners can read from and write to the alternate index.

SPACE — (SP) specifies the MUSIC primary space allocation, in units of 1024 bytes (K) for the data or index component.

TRACKS — (TRK) specifies the primary and secondary space allocations in terms of 3330 disk pack tracks (twenty 512-byte blocks per track).

UNIQUEKEY — (UNQK) specifies for alternate index keys that repeating values are not allowed in the base cluster.

UPGRADE — (UPG) specifies that the alternate index being defined is to be placed in the upgrade set of the associated base cluster.

WRITE — (WR) specifies that the alternate index is to be given the WSHR attribute, meaning that non-owners can read from and write to the alternate index.

DEFINE ALTERNATEINDEX Examples

Example 1: Defining an alternate index, specifying the alternate key position as length 15 and offset 5, and the related base cluster "BASE.ESDS". Note that "BASE.ESDS" must previously have been defined.

```
/INCLUDE AMS
DEFINE ALTERNATEINDEX                                  -
            (  NAME(AIX1.DAT)   RELATE(BASE.ESDS) -
               KEYS(15 5) )
```

Example 2: Defining an alternate index specifying that key values can repeat (NONUNIQUEKEY) and that when changes are made to the base cluster, they should not be reflected in the alternate-index.

```
/INCLUDE AMS
DEFINE AIX                                             -
(NAME(AIXFILE) REL(BASEFILE) NOUPGRADE NONUNIQUEKEY)
```

DEFINE CLUSTER

Usage

```
DEFINE  CLUSTER
          (NAME(entryname)
          [CYLINDERS(primary[ secondary])|
             RECORDS(primary[ secondary])|
             TRACKS(primary[ secondary])]
          [CONTROLINTERVALSIZE(size|4096)]
          [FREESPACE(CI-percent[ CA-percent]|0 0)]
          [INDEXED|NONINDEXED|NUMBERED]
          [KEYS(length offset|64 0)]
          [PRIVATE|COMMON|SHARE|PUBLIC|WRITE
          [RECORDSIZE(average maximum|4089 4089)]
          [REPLACE]
          [SPACE(primary|40)]
          [SECSPACE(secondary|0)])


        [DATA
          ([CONTROLINTERVALSIZE(size|4089)]
          [CYLINDERS(primary[ secondary])|
             RECORDS(primary[ secondary])|
             TRACKS(primary[ secondary])]
          [KEYS(length offset)]
          [RECORDSIZE(average maximum)]
          [SPACE(primary|40)]
          [SECSPACE(secondary|0)])]

        [INDEX
          ([CONTROLINTERVALSIZE(size|512)]
          [CYLINDERS(primary[ secondary])|
             RECORDS(primary[ secondary])|
             TRACKS(primary[ secondary])]
          [NAME(entryname)]
          [SPACE(primary|10)]
          [SECSPACE(secondary|0)])]
```

The DEFINE CLUSTER command creates a VSAM cluster. This consists of a data component, and in the case of a key sequenced data set, an index component.

User Data Set files are supported only for the data components of base clusters. A base cluster with a UDS for the data component may not have alternate indexes built over it (this restriction is enforced simply by not recognizing the UDS specification in the DEFINE ALTERNATEINDEX command).

To use a UDS as the data component of a base cluster, you must include a /FILE statement that defines the UDS file. In the the NAME keyword of the DATA clause, instead of the file name, you specify "/FILE#", where "#" is the unit number of the /FILE statement that defines the UDS file (for example, /FILE1). Note that a valid cluster name may still be included in the NAME parameter in the CLUSTER clause. This will then determine the form of the name for the index component, if that name is not explicitly specified.

Keywords that occur in both the CLUSTER clause and the DATA clause apply only to the DATA clause. These keywords need not be specified in both clauses. If a keyword appears in both clauses, the keyword in the DATA clause is used. The reason for this is that if AMS command streams from other systems are used, the specification intended for the data component will be used for the data component.

Abbreviation: DEF CL

The only required parameter is the NAME subparameter in the CLUSTER keyword. The following describes each of the parameters.

COMMON (COM) specifies that the file names for this VSAM object are to be placed in the common index

CONTROLINTERVALSIZE
(CISIZE) specifies the controlinterval size to be used for the DATA and/or INDEX component of the VSAM file (default: 4096 for data component, 512 for index component)

CYLINDER (CYL) specifies the allocation of space for the data component in terms of cylinders of a 3330 disk pack (this is 19 tracks of 20 512-byte blocks each)

FREESPACE (FSPC) specifies the freespace percentage for control intervals. Note that although the control area freespace percentage is scanned, currently it is ignored.

INDEXED (KSDS) specifies that the VSAM cluster is a key sequenced data set (KSDS). Note that this is the default

KEYS specifies the length and offset of the key of the VSAM object. Note that the offset is relative to 0, so that if the key starts in column 1, the value that you provide should be 0.

NAME (NA) specifies the name of the VSAM object. When used in the CLUSTER keyword, it specifies both the name of the VSAM data component and the default root for the index component. Note that the data component name is the one used when opening the file.

In the special case of a name specified in the CLUSTER keyword of "name.DAT", Access Method Services provides a default name for the index component of "name.IDX". Otherwise, Access Method Services appends ".IDX" to the data component name by default.

NONINDEXED (ESDS) specifies that the VSAM cluster is an entry sequenced dataset. Note that in this case, no index component exists.

NUMBERED (RRDS) specifies that the VSAM cluster is a relative record data set.

PRIVATE (PRIV) specifies that the VSAM objects are to be given a file attribute of PRIVATE, and therefore cannot be shared with other users (this is the default).

PUBLIC (PUBL) specifies that the VSAM objects are to be given the file attributes of COM and SHR, meaning that non-owners can read from the cluster and need not prefix the file name with the owner's userid.

RECORDS (REC) specifies the space allocated to the data component in terms of the maximum logical recordsize (default: CISIZE - 7) and a secondary allocation in terms of the

maximum logical record size.

RECORDSIZE — (RECSZ) specifies the average logical recordsize and the maximum logical recordsize for the data component. Note that the average logical recordsize is scanned but ignored.

REPLACE — (REPL) specifies that if the VSAM objects being defined already exist, they are replaced (note that this is NOT the default; in other words, if the VSAM objects being defined exist, the DEFINE command terminates with an error condition).

SECSPACE — (SECSP) specifies the MUSIC secondary space allocation in units of 1024 bytes (K) for the data or index component. If 0 is specified (the default), this means that a secondary space allocation will be 50% of the current file size.

SHARE — (SHR) specifies that the VSAM objects being created are to be given the SHR attribute, meaning that non-owners can read from the cluster.

SPACE — (SP) specifies the MUSIC primary space allocation, in units of 1024 bytes (K) for the data or index component.

TRACKS — (TRK) specifies the primary and secondary space allocations in terms of 3330 disk pack tracks (twenty 512-byte blocks per track).

WRITE — (WR) specifies that the cluster is to be given the WSHR attribute, meaning that non-owners can read from and write to the cluster.

In general, this attribute is not recommend for the average application because it places heavy responsibility upon the application program to guarantee the integrity of the data.

DEFINE CLUSTER Examples

Example 1: Defining a cluster using all of the defaults. Note that this is not recommended since the default key size and offset (64 and 0) is unlikely to meet the requirements of your application.

```
/INCLUDE AMS
DEFINE CLUSTER                          -
            (NAME(BASEFILE.DAT)    )
```

Example 2: Defining a key-sequenced cluster, specifying the space in terms of K bytes (multiples of 1024 bytes), that it should replace any existing file of the same name, that the maximum recordsize should be 80 bytes, that the controlinterval size for the index component should be 1024 bytes, that the files created should be SHR (other users can access them), and that the keysize and offset should be 20 and 10, respectively. Note that the name of the index component defaults to: "BASEFILE.IDX".

```
/INCLUDE AMS
DEFINE CLUSTER                                          -
            (NAME(BASEFILE)  REPLACE SHR  )             -
            DATA( RECORDSIZE(1 80)  KEYS(20 10)  )      -
            INDEX( CONTROLINTERVALSIZE(1024)  )
```

Example 3: Defining a key-sequenced dataset, specifying explicit names for both the data component and index components.

```
/INCLUDE AMS
DEFINE  CLUSTER ( INDEXED         )       -
         DATA    ( NAME(DATAFILE)  )       -
        INDEX    ( NAME(INDEXFILE) )       -
```

Example 4: Defining a relative record data. Note that the abbreviations KSDS, ESDS, and RRDS may be used for INDEXED, NONINDEXED, and NUMBERED, respectively. The space here is specified in tracks of a 3330 disk, which is converted into a space allocation in the MUSIC/SP file system.

```
/INCLUDE AMS
DEFINE CLUSTER ( NAME(BASEFILE.DAT)   RRDS )   -
       DATA    ( TRACKS( 10 1) )
```

DEFINE PATH

Usage

```
DEFINE  PATH
          (NAME(entryname)
          PATHENTRY(entryname)
          [PRIVATE|COMMON|SHARE|PUBLIC|WRITE]
          [REPLACE])
```

The DEFINE PATH command creates a path between an alternate index and its associated base cluster. The only required parameters are the NAME and the PATHENTRY subparameters of the PATH keyword.

Abbreviation: DEF PATH

The following describes each of the parameters.

COMMON — (COM) specifies that the path filename is placed in the common index

NAME — (NA) specifies the name of the VSAM path. This parameter is required.

PATHENTRY — (PENT) specifies the alternate index which is used to access the base cluster. This parameter is required. Note that this file must have been previously defined.

PRIVATE — (PRIV) specifies that the path is to be given a file attribute of PRIVATE, and therefore cannot be shared with other users (this is the default)

PUBLIC — (PUBL) specifies that the path is given the file attributes of COM and SHR, meaning that non-owners can read from the file, and need not prefix the file name with the owner's userid.

REPLACE — (REPL) specifies that if the path being defined already exists, it is replaced (note that this is NOT the default; in other words, if the path being defined exists, the DEFINE PATH command will be terminated with an error condition)

WRITE — (WR) specifies that the path is to be given the WSHR attribute, meaning that non-owners can read from and write to the file.

DEFINE PATH Example

Defining a path, specifying that it is to be public. Note that all associated files (base cluster and alternate indexes) need not be public for access to be successful, but they must at least be SHR (for write access they must be WSHR).

```
/INCLUDE AMS
DEFINE PATH ( NAME(DATA.PATH1) PATHENTRY(BASE.AIX1) PUBL)
```

DELETE

Usage

```
DELETE  entryname|[(entryname ... entryname)]
         CLUSTER|ALTERNATEINDEX|PATH|NONVSAM
```

The DELETE command deletes a file (VSAM or non-VSAM). A single positional parameter specifying the name of the file to be deleted must be specified.

Abbreviation: DEL

The following describes each of the parameters.

entryname — specifies the name of the VSAM file to be deleted. If no other parameter is specified, then an attribute of CLUSTER is assumed.

If the file to be deleted is a VSAM cluster, then the following occurs. For each alternate index associated with the cluster, all paths are deleted, and then the index and data components of the alternate index are deleted. When all alternate indexes have been deleted, then the index and data components of the base cluster are deleted.

If the file to be deleted is a VSAM alternate index, then the following occurs. All paths are deleted, and then the index and data components of the alternate index are deleted. The name of the alternate index is removed from the base cluster.

If the file to be deleted is a VSAM path, then it is deleted. The name of the path is removed from the associated alternate index.

If the file to be deleted is a non-VSAM file, then it is opened and deleted. Note that by saying NONVSAM, you mean that the file is not a component of a VSAM cluster. If the file happens to be a VSAM object, the delete will not be performed. Thus if for some reason only a portion of a VSAM cluster needs to be deleted, the AMS DELETE command cannot be used. In this case, use the MUSIC/SP PURGE command in *Go mode.

CLUSTER — (CL) specifies that the entryname refers to a VSAM cluster. This parameter is the default.

ALTERNATEINDEX — (AIX) specifies that the entryname refers to an alternate index. This parameter is optional.

NONVSAM specifies that the entryname refers to a non-VSAM file. This parameter is optional.

PATH specifies that the entryname refers to a path. This parameter is optional.

DELETE Examples

Example 1: Deleting two VSAM clusters. Note that all associated files will be deleted as well.

```
/INCLUDE AMS
DELETE ( XXX1 XXX2 )    CLUSTER
```

Example 2: Deleting a VSAM alternate index. Note that paths defined over this alternate index will be deleted as well.

```
/INCLUDE AMS
DELETE  XXX1.AIX        ALTERNATEINDEX
```

Example 3: Attempting to delete a non-VSAM file and a VSAM cluster within the same command. In this case, only the non-VSAM file will be deleted.

```
/INCLUDE AMS
DELETE   (XXX.KSDS   XXX.NONVSAM)  NONVSAM
```

LISTCAT

Usage

```
LISTCAT   ENTRIES(entryname) -
          [ ALL ]
```

The LISTCAT command displays the attributes of a VSAM file. The names of files that are logically associated with this file are reported. Physical attributes for both the DATA and INDEX components (where applicable) are reported.

Abbreviation: LISTC

The following describes each of the parameters.

ENTRIES specifies the name of a VSAM object whose attributes are to be displayed.

ALL specifies that all information pertaining to the object specified by ENTRY is to be displayed. The default is that only the associated files are listed.

LISTCAT Examples

Example 1: Listing the type of VSAM cluster as well as the names of any associated files. No other attributes will be listed.

```
/INCLUDE AMS
```

```
LISTCAT ENTRIES( XXX.CLUSTER )
```

Example 2: Listing all of the information available for the VSAM cluster.

```
/INCLUDE AMS
LISTCAT ENTRIES( XXX.CLUSTER )
```

REPRO

```
REPRO     INDATASET(entryname)|INFILE(ddname)
          OUTDATASET(entryname)|OUTFILE(ddname)
          [ COUNT(number) ]
          [ NOREPLACENONVSAM ]
          [ RELEASENONVSAMSPACE ]
          [ SKIP(number) ]
```

The REPRO command copies one file to another. The function performed is essentially that of the MUSIC/SP COPY command. The two commands are different in that the REPRO command performs VSAM I/O whenever a dataset is a VSAM file.

The COPY command is recommended for copying non-VSAM files whenever you are in *Go mode. When in AMS, the REPRO command may be used with similar results. Note that the REPRO command does not prompt for permission to replace existing non-VSAM files.

The INDATASET parameter must name an existing VSAM or non-VSAM file containing data. Note that a VSAM PATH file is not valid. As well, specifying the name of an index component of a KSDS or an alternate index is invalid. (To copy these datasets separately, use the MUSIC/SP COPY command.)

The OUTDATASET parameter is required. If a VSAM file is specified, a valid base cluster (KSDS, ESDS or RRDS) file must be named.

Abbreviation: REP

The following describes each of the parameters.

COUNT — specifies the number of logical records that are to be written to the output file. After this number of records has been written to the output file, the REPRO operation is terminated.

INDATASET — (IDS) specifies the name of a VSAM base cluster or non-VSAM dataset to be copied to the output file.

INFILE — (INF) specifies a DDNAME which is used to obtain the input. This is useful if reading from a UDS file.

OUTDATASET — (ODS) specifies the name of a VSAM base cluster or non-VSAM file into which the contents of the input file are to be copied.

OUTFILE — (OUTF) specifies a DDNAME which is used to write the output from REPRO. This is useful if the output file is a UDS file.

NOREPLACENONVSAM

specifies that REPRO is not to replace a non-vsam target file. Without this parameter, the file is replaced if it exists.

RELEASENONVSAMSPACE
specifies that unused space in the non-vsam target file is to be released when the file is closed. The default is not to release unused space.

SKIP specifies the number of logical records from the input file that are skipped before starting to write records to the output file.

REPRO Examples

Example 1: Copying input data into an output file. Note that REPRO automatically determines whether the input and output files are VSAM or non-VSAM, and therefore this information is not needed.

```
/INCLUDE AMS
REPRO INDATASET( INPUT.DATA ) OUTDATASET( OUTPUT.DATA )
```

Example 2: Copying input file to output file, skipping the first 100 input records and limiting the number of records written to the output file to 500.

```
/INCLUDE AMS
REPRO                                  -
      IDS( INPUT.DATA ) SKIP (100)     -
      ODS( OUTPUT.DATA) COUNT(500)
```

Archiving and Retrieving User Files

Note: The FILARC, FILRST, and FILCHK utilities described later in this chapter are the recommended programs to be used for dumping and restoring files, and they should be used rather than ARCHIV and RETREV. FILARC can dump any type of file, and does so faster and more compactly than ARCHIV. FILRST can restore directly to the Save Library. ARCHIV and RETREV are included in this manual for the benefit of those users who may still have dump tapes created by ARCHIV, since such tapes are not compatible with FILRST.

The user can cause all his own files to be copied to a magnetic tape by using the ARCHIV program. The referenced files are not altered by this copy operation. Options are available to copy only those files that have not been used for some time. Each file is written on the tape in the form of a /SAVE job so that it can be reloaded at a later time. This program features the ability to write to two tapes simultaneously thus giving you two complete copies of your files on the two tapes. (To archive User Data Sets (UDS) files, refer to the writeup on the utility UDSARC.)

To retrieve a file or a set of files from the tape, you can use the RETREV program.

The user should note that both the archive (ARCHIV) and the retrieve (RETREV) programs are run from batch since they use magnetic tape.

ARCHIV Program

Files with record format U (undefined format) will not be copied to tape. This would be the case if, for example, the file is a VS APL workspace. Files with record length longer than 80 bytes will be truncated to 80 bytes.

The following illustrates the control statements set up for the ARCHIV program:

```
/FILE n TAPE ... etc.
/INCLUDE ARCHIV
.......   parameter statement (see below)
```

Parameter Statement:

```
TAPE=n[,PURGES=n][,UDATE='ddmmmyy']
```

TAPE=n is the MUSIC I/O unit number to be used. n must be 1 or 2 for a magnetic tape, or 7 for the card punch. If two tapes are to be written, the control statement would be TAPE=1,2 (up to two unit numbers can be specified on the control statement). A /FILE statement must be present for each tape to be used. Note that the record size (RSIZ) must be specified as 80 and the blocksize (BLK) must be a multiple of 80. A sample /FILE statement is given below:

```
/FILE 1 TAPE BLK(800) RSIZ(80) VOL(mylib)
```

PURGES=n Causes a /PURGE job to be written on unit n (normally 7) for each file which is archived. This facilitates the purging of archived files. (Note: only the MUSIC Systems Administrator is normally authorized to run purge jobs from batch.)

UDATE='ddmmmyy'
Causes only those files with the last used date of the specified date or earlier to be archived. The date is given as a 7-character day, month, year string such as UDATE='01JAN74'.

Retrieving Files: RETREV

Files can be retrieved from the tape prepared by the ARCHIV program and punched on cards or written on a magnetic tape or disk data set by running the following batch program.

```
/FILE statements as required
/INCLUDE RETREV
... parameter statement  (see below)
... list of file names starting in col 1, 1 per card
```

Parameter Statement:

```
INPUT=n[,OUTPUT=m][,PRINT][,SELECT='ALL']
```

INPUT=n This parameter specifies the MUSIC I/O unit number to be used to read the archive tape. Unit 1 is assumed if INPUT=n is not specified. A /FILE statement defining the tape must be present and its unit number must be *n*. You must make sure that the BLK parameter on this /FILE statement matches the one used when the tape was created. The RSIZ must be specified as 80 on the /FILE statement.

OUTPUT=m This parameter gives the MUSIC I/O unit number where the retrieved files are to be written. The default of OUTPUT=0 is taken to mean that no output is to be generated. To punch the files, use OUTPUT=7. If you just want the files to be printed, you should use the PRINT option documented below.

PRINT This option causes all retrieved files to be printed. Normally the retrieved files are not printed. This option is independent of the OUTPUT option. For example, you can say OUTPUT=0,PRINT to just print the files.

SELECT='ALL' This option will cause all the files on the input tape to be retrieved. The list of file names will not be required in this case.

Debug Facility

The Debug facility enables you to debug programs running on MUSIC at the machine language level. You can scroll up and down through main storage. The Program Status Word, any of the 16 general purpose registers, and main storage can all be modified. You can single step through your program or set breakpoints and run continuously. You can set watchpoints and cause your program to be stopped when they are altered.

Debugging a program consists of being able to monitor the execution of the program, controlling the input to the program, observing the results, and possibly altering memory while the program is in progress to affect the manner in which it executes.

Debugging at the machine language level gives you complete access to the machine resources. A disadvantage, of course, is that high-level languages that have been translated to machine language by compilers become unrecognizable. Many compilers, however, have the capability of displaying the machine language code that they are generating. You may find that Debug is most effective when used to debug Assembler language programs. An exception is VS/FORTRAN. Debug runs co-operatively with TESTF, a VS/FORTRAN debugger. In this mode, Debug displays the current VS/FORTRAN statement being executed in the message area. For more information, refer to the topic entitled "TESTF - MUSIC/SP VS/FORTRAN Debugger" in *Chapter 8. Processors*.

Another function of Debug is to give you executable access to MUSIC's main memory and registers. With this, you can look at various locations in memory, and set up sample instructions to discover how they work. Debug provides an excellent vehicle to teach, and learn about, /370 architecture.

Some of the facilities of Debug (described below) are: single-stepping your program instruction by instruction: altering memory and values in both the general purpose and floating point registers; altering the Program Status Word; searching for hexadecimal or character strings in memory; setting break-points (places where your program is halted and control is returned to the Debug facility); entering TRACE mode (where you receive an instruction-by-instruction tracing of your program's execution); and setting watchpoints (monitored storage that causes your program to be interrupted when it changes).

Keep in mind that some of MUSIC's main storage is read-protected. Thus some memory locations are inaccessible to you, unless you have the required high-level privileges. (These are granted only to the systems programmers at your site for necessary systems work.) Write-protected storage cannot be altered, unless you again have the required high-level privileges. For example, you cannot change anything below location X'1000' (hexadecimal 1000). Although you can type over these memory locations in the memory display, your changes are not applied to the actual storage locations.

How Debug Works

To understand how to use Debug, and appreciate its limitations, you may wish to understand a little bit about how Debug does what it does. If not, then skip to the next section.

A Supervisor Call (SVC) instruction, in the /370 architecture, causes a transfer to the Operating System, and usually the execution of some function or module within the operating system. This is done via the mediation of both the /370 hardware and the control program (in this case, MUSIC/SP). An SVC may involve a switch to supervisor state from problem state, or it may not. Generally speaking, your program interfaces to the Operating System (MUSIC/SP) via these Supervisor Call instructions, (SVCs).

Debug uses a special SVC defined in the MUSIC system. When you single-step your program, Debug inserts this SVC just after the instruction to be executed, remembering what was there previously. Then, it

transfers control to the instruction to be executed. Your instruction is executed normally, and then following that, the Debug SVC instruction is executed. When this Debug SVC is executed, Debug re-gains control, and re-displays the memory display panel, after restoring the contents of memory where the Debug SVC was placed.

When a conditional BRANCH instruction is to be single-stepped, there are two possible target addresses: when the BRANCH is "taken" (successful), and when it is not. Debug checks both of these addresses and places a Debug SVC at both locations, so that when your program reaches one of them, it will re-gain control.

Recalling that some of MUSIC's main storage is Write-protected, you can appreciate that if one of the possible targets of a BRANCH instruction is in protected storage, then Debug cannot alter that location to place a Debug SVC there. Consequently, Debug issues a message stating that the branch location is in protected storage. At this point, you cannot single-step your program any more, and must issue the GO command.

How can you re-gain control after issuing the GO command? There are two ways: first, by pressing PA1 when Debug is executing and second, by placing "break-points" within your program.

In the first case, when the execution of your program is suspended by MUSIC to give other users of the system a chance to do some work, it will not be given control again at the place where it was interrupted. Instead, Debug will gain control. Effectively, you have "broken out" of whatever your program was doing by pressing PA1, and re-entered Debug. This can be useful in the case of so-call "infinite" loops, or similar circumstances.

In the second case, you must plan ahead. You set break-points using the BREAK command (see below). You indicate in this command an address that Debug will remember (and display in the module map on the memory display panel). Whenever you issue the GO command to Debug, it takes all of the break-points from its list and places Debug SVCs at those locations, remembering the previous contents of memory. Then, whenever control reaches one of those locations, the Debug SVC is executed, and Debug re-gains control.

Invoking Debug

Debug can be invoked in several ways, depending what you want to do with it. It can be invoked in "stand-alone" fashion, or as an option on certain /LOAD processors. You may have a program that you wish to debug using Debug, or you may simply wish to gain access to MUSIC main storage and registers.

Command Line Invocation

Simply typing Debug at the *Go prompt will start up the facility with the memory display initially positioned at hexadecimal location x'4800'. Obviously, this is arbitrary, and you can display any other location in memory to which you have access.

To Debug a program in object module format, you can specify a MUSIC file name as a parameter. For example,

```
debug  program.obj
```

will invoke Debug with the program contained in "PROGRAM.OBJ" loaded into memory. The memory display will be positioned at the first Control Section (CSECT) found in the object module.

There are additional parameters that you can specify in order to further control how Debug loads your program. These are keyword parameters (not dependant upon the order on the command line) which require

a value. Specify the value in parentheses, like this: PARMS(hello). Note that to specify parameters without specifying an object module, type a period (".") in place of the object module name. For example:

```
debug . parms(hello)
```

The parameters for Debug are:

EPNAME — the CSECT (entry-point) name where the memory display should be positioned. Note that this is where the program will be started when you issue a STEP or a GO command, so be careful when you specify the name (you can change the starting address manually within Debug, of course).

PARMS — a parameter string that will be passed to the program via an address in register 1. Note that this follows standard calling conventions

REGION — a numerical value which is the region size to be used when your application program is loaded with Debug. Sometimes for larger programs, a region size that is larger than the size normally used by your program alone is required, since the code for Debug resides in the user region as well.

FILES — a MUSIC file name containing /FILE definitions to be used by your program. Debug scans this file and extracts /FILE Job Control Statements at the beginning of the file. Other control statements are not used, but /FILE statements are replicated when your program is loaded with Debug.

TRACE — a MUSIC file name which is used to capture output on unit 6; specifically, output from instruction tracing. This enables you to debug full-screen applications more easily, as well as providing an easy way to view long trace sequences.

Invoking XMON

OS mode load modules can be debugged "as-is" by specify Debug as an option on the parameter statement, when using /LOAD XMON. An example follows:

```
/FILE LMOD NAME(PROGRAM.LMOD) SHR
/LOAD XMON
PGMNAME  LMOD   DEBUG
```

Invoking Debug from /SYS

Usually when your program is executed, quite a number of other modules have already been loaded into your user region and called in the process of starting and running your job. However, for some reason you may wish to give Debug control as the the first user program in your user region.

You can do this by specifying a /SYS statement like this:

```
/SYS Debug
```

This tells MUSIC/SP that Debug is to be brought into memory and given control before any other program gains control in the user region. Note that your user region has already been set up for execution to commence at the first instruction (usually at hexadecimal location x'1000'). Keep in mind that if you are running a compiler, not only has the compiler not been started but OS simulation has not even begun.

In most cases, this method of invoking Debug is not very useful, as too much remains to be done before your

application program gets loaded into memory.

The command "DBG" in *Go mode uses /SYS DEBUG. Running DBG causes Debug to be brought into memory very quickly. It is equivalent to typing "DEBUG" except for the start-up speed.

Invoking a Compiler

When using a compiler you can Debug your program by specify Debug as an option on the /JOB statement. After your program is compiled and loaded into memory, but prior to the execution of your program, Debug is invoked. An example follows.

```
/LOAD ASM
/JOB GO,Debug
IEFSRBR  CSECT
         SR   15,15
         BR   14
         END
```

Screen Display

The following diagrams illustrate the two major display modes of Debug. F4 (Flip) is used to go from one display to the other.

```
--------------------------- Debug Facility  ------------------------------
Command ==> _
Program Status Word:FF250000 40004800  Condition Code:00   Prev Addr:000000

R0:00000000 00005334 00000000 00000000 00000000 00000000  00000000 00000000
R8:00000000 00000000 00000000 00000000 00000000 00005500  00004A60 00004800

 Instruction: STM     Oper 1: 000000 Oper 2: 00482C
004800  :     900FF02C 4100F00C 47F0F014 C4C5C2E4  : ..0. ..0. .00. DEBU :
004810  :     C7404040 1B110A08 18EF18F0 980DE02C  : G    .... ...0 q.\. :
004820  :     50F0E028 47F0EO70 000048C8 000025D0  : &0\. .0\. ...H ...] :
004830  :     00000C1C 000048C7 00080000 000025D0  : .... ...G .... ...u :
004840  :     00080000 000048C7 00004800 00001120  : .... ...G .... .... :
004850  :     0000466C B08813AC 00880850 00000000  : ...% .h.. .h.& .... :
004860  :     000042D8 400041C0 00004800 00000000  : ...Q  ..[ .... .... :
004870  :     1B1105EF 0AFF0000 00000000 00000000  : .... .... .... .... :

1                    :                 :                :
5                    :                 :                :
9                    :                 :                :
13                   :                 :                :
-------------1:Help 3:End 7:UpMemory 8:DownMemory 4:Flip PA1:Watch-–--------
  2:Run  5:StepPast 12:Step  10:PtData 11:PtInst  9:SetBreak  6:PrevScr
```

Figure 10.2 - Debug Memory Display

```
-------------------------- Debug Facility  ------------------------------
Command ==> _

Program Status Word:FF250000 40004800  Condition Code:00   Prev Addr:000000

R0:00000000 00005334 00000000 00000000 00000000 00000000  00000000 00000000
R8:00000000 00000000 00000000 00000000 00000000 00005500  00004A60 00004800

    Address   Instruction  Storage         Operand 1   Operand 2
---> 004800      STM        900FF02C        000000      00482C
     004804      LA         4100F0DC        000000      00480C
     004808      BC         47F0F014        00000F      004814
     00480C      ????       C4C5
     00480E      ????       C2E4
     004810      ????       C740
     004812      STH        40401B11        000004      005E45
     004816      SVC        0A08            000000      004808
     004818      LR         18EF            00000E      00480F
brk> 00481A      LR         18F0            00000F      004800
     00481C      LM         980DE02C        000000      004A8C
     004820      ST         50F0E028        00000F      004A88
     004824      BC         47F0E070        00000F      004AD0
-------------1:Help 3:End 7:UpMemory 8:DownMemory 4:Flip PA1:Watch-=--------
  2:Run  5:StepPast 12:Step  10:PtData 11:PtInst  9:SetBreak  6:PrevScr
```

Figure 10.3 - Debug Instruction Display

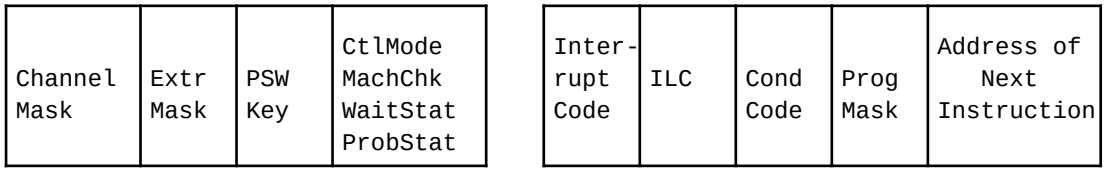
COMMAND: Command Line

The Command Line field is used to enter Debug commands, as well as MUSIC commands. MUSIC commands are distinguished from Debug commands by prefixing them with a slash. Thus, DISPLAY F will display memory at hexadecimal location 00000F, but /DISPLAY F will attempt to display a file called "F" on the screen.

PSW: Program Status Word

The Program Status word is used in /370 architecture to keep track of the address of the next instruction, as well as various flags such as the condition code, etc.

Program Status Word



Bit Positions

When a program interrupt occurs, usually the address in the PSW points to the address of the instruction FOLLOWING the offending instruction. Thus the Instruction Length Count (ILC) can be used to determine how far backwards to step. The ILC is 1 for a 2-byte instruction, 2 for a 4-byte instruction, and 3 for a 6-byte instruction.

Condition Code Field

This field displays the condition code found in the PSW (Program Status Word).

The condition code is set by a variety of /370 instructions, particularly those that are intended to perform comparisons. Generally speaking, the condition code reflects the completion status of the instruction that set it. Not all instructions set the condition code, and when one of these instructions are executed, the condition code remains unchanged.

Instructions that operate on the condition code settings are typically BRANCH instructions. Within these instructions, there is usually a mask field that corresponds to the various condition code settings.

Previous Address Field

This field displays the previous address that was known to Debug. Various Debug commands cause this field to be updated. Note that it is not necessarily the address of the previous instruction. However, it it is sometimes useful, as the POINT command can be used on this field to re-position the memory display.

General Purpose Registers

These fields display the contents of the 16 general purpose registers. These registers are used for arithmetic, addressing, comparisons, etc. You can change the values in the registers. You can use values in the registers to "point" (see below) to memory to be displayed. As well, the value in a register can be treated as an address, and a "break-point" can be assigned to that address using a function key.

When your program is given control the contents of the registers will be set to the updated values. Any address calculations performed by various instructions are displayed when the instruction is displayed, as in the Instruct Display mode. Note that R0 is never used in an address calculation.

Memory Display Partially Decoded Instruction

This field displays an instruction that has been partially decoded. The operation code is displayed, and operands 1 and 2 are shown. Whenever the operand is a register, then register number is displayed. When the operand is an address reference, then the computed address is displayed.

Note that the displayed address prior to execution may not be the one that is actually used. The address that is actually used depends upon the contents of any registers involved in the address calculations, and if these values change, then of course the address will, too.

Memory/Character Display

These fields are used to display main storage and the character equivalent of main storage. Each line represents 16 bytes of main storage, with 8 lines in total. Thus there are 128 bytes of storage represented on the display.

The column of hexadecimal numbers on the left-hand side represents the starting address for each line of data. The 16 bytes are broken up into words of 4 bytes each. The right-hand side of the display shows the EBCDIC character equivalents of the hexadecimal information in the middle. This, too, is broken up into 4 character chunks for easy alignment.

The location of the next instruction is highlighted, if it is within the 128 bytes of main storage displayed on the screen. This instruction is determined by the value in the PSW.

Note: You cannot modify the character representation of the data, but you can modify the hexadecimal representation. Only valid Hex digits are acceptable. If you are viewing memory that you are not allowed to modify, any changes that you make to the screen will be ignored.

Map Table Entries

These fields display the Map Table entries. Each entry is a name/address pair that can be used with the Point function key to position the Memory Display to that address.

Entries can be made in several different ways. First, at start-up time, symbols known to the Loader are placed into the Map Table. Running from object modules results in all external symbols being placed into the Map Table, while using the Debug option with XMON results in only the main entry point being placed into the Map Table.

Second, whenever modules are loaded using the LOAD SVC, their name and address are placed into the Map Table. This is done automatically by Debug.

Third, you can make explicit entries into the Map Table with the ENTRY command.

Floating Point Registers

These fields display the contents of the 4 floating point registers. These registers are used for floating point arithmetic exclusively, and since most machine language programs rarely use them, their display is optional. They are displayed in place of the Map Table, when the FLOAT command is issued.

As with the general purpose registers, you can alter the values in the floating point registers. When your application program resumes, the updated values will be placed in floating point registers.

Note that whatever values are placed here are NOT normalized. In order to do this, you must input them as such, or use an instruction that does this, if you so desire.

Instruction Display

These display partially disassembles instructions and displays them, one per line. As well, operands 1 and 2 are displayed. In the case of a storage to storage instruction, the source and target addresses of the instruction are displayed. You can alter the instruction, and pressing ENTER will display the new instruction. This display mode is ideal for learning about instructions at the machine level, and stepping through a program instruction by instruction.

An arrow points to the next instruction to be executed. If a breakpoint is visible on the display, it is indicated by "brk>" (see figure 10.3).

Other Topics

Hex Addresses and Numbers

Hexadecimal numbers consist only of the hexadecimal digits 0 through 9 and A through F. Using any other characters in a hexadecimal parameter constitutes an error. Leading zeros need not be used. Floating point and general purpose registers must, however, be completely filled with hexadecimal digits. This also applies to memory fields. No leading or embedded blanks are acceptable.

Hexadecimal addresses have a maximum length of 6 digits, due to the 370 architecture. Examples:

```
        0b312f and 000100   are valid addresses
   but  0x0359   is not valid: 'x' is not a hexadecimal digit
        0067359  is not valid: it is greater than 6 digits in length
```

Commands where hexadecimal parameters are used: Break, Delete, Display, Entry, Find, and Go.

Calculator Function

Debug has a built-in hex calculator function. As well as maintaining a separate arithmetic symbol table, it will search the Map Table, and use any symbols there in calculations. Addition, subtraction, multiplication and division are supported. Decimal to hexadecimal conversion can also be performed. You can use constants or symbols, and can create new symbols. The Program Status Word and registers are automatically entered in the calculator symbol table, so that you can use them in calculations. (Note that only the address portion of the PSW is used.)

You can either use the CALC command (described below) or simply enter an arithmetic expression. If Debug doesn't recognize text entered on the command line as a command, it checks to see if it is a valid arithmetic expression or assignment statement. If so, it will perform the indicated operation.

This function can be extremely handy. You can calculate an address and store it. With a number of commands that refer to memory, you can use a stored symbol instead of a hexadecimal constant.

PI: Program Interrupts

Program interrupts are error conditions that the machine cannot handle. For example, an operation exception means that the operation code wasn't recognized by the processor. Here are the program interrupts that Debug recognizes:

01- operation exception
02- privileged operation exception
03- execute exception
04- protection exception
05- addressing exception
06- specification exception
07- data exception
08- fixed point overflow exception
09- fixed point divide exception
10- decimal overflow exception
11- decimal divide exception
12- exponent overflow exception
13- exponent underflow exception
14- significance exception
15- floating point divide exception

The causes of PIs are sometimes easy to determine, and sometimes difficult Usually, a protection exception is easy, since the instruction obviously references protected memory. Debug displays the offending instruction. However, an operation exception may be more difficult, as you may have branched to location 0 (usually a bad address in your program).

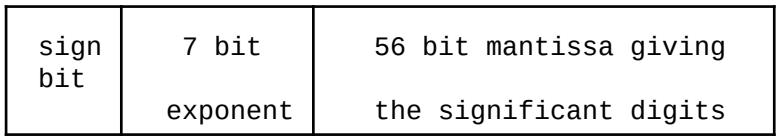
Format of Integer Numbers

The general purpose registers are used to store the 2's complement representation of integer numbers. On /370 computers, integers are represented in a binary code that incorporates the sign of the number. The leading bit is termed the "sign bit" and is 0 for a non-negative number and 1 for a negative number. The remaining 31 bits determine the absolute value of the number, in 2's complement.

For example, the number "-1" is represented in hexadecimal by "FF FF FF FF", while the number 1 is represented by "00 00 00 01".

Format of Floating Point (Real) Numbers

The floating point registers are used to store the binary representation of real numbers. On /370 computers, real numbers are represented in a manner akin to scientific notation, whereby the significant digits are represented in fixed precision and the position of the decimal point is determined by an exponent.



If the number is negative, the sign bit is 1; otherwise it is 0. The 7-bit exponent is a biased exponent with 64 or x'40' being the zero point. When the exponent is x'40' then no shift in the position of the decimal point occurs. For example, 40 10 00 00 00 00 00 00 represents the number '1'.

The mantissa is usually normalized leftwards to the nearest hex digit although there exist instructions which perform unnormalized arithmetic.

Command Files

A Debug "command file" is simply a file containing Debug commands. Debug can read these files and execute the commands automatically. Debug can have an initial command file for running at start-up and other command files can be executed during debugging.

Comments are denoted by an asterisk in the first column. Debug commands can be in either upper or lower case. MUSIC commands may be issued as well.

Although you cannot change the memory display in this mode, you can assign values to registers (eg. R2 = AOA), change the PSW, and alter memory via the REP command (see below).

Initial Command File

When Debug starts up, it checks for a /FILE statement with a ddname of "DBGCMD". If found, commands read from this file are executed prior to passing control to the user. Thus, this file can be used to perform a number of repetitive tasks prior to the user gaining control.

Command Files

Debug can execute a file containing Debug commands as well as executing them at start-up. At start-up, if Debug finds a DDNAME "DBGCMD", it reads this file and executes each command, before passing control to the user. With the command file facility, you can cause Debug to go out to a MUSIC file and dynamically execute the commands contained in that file. You instruct Debug to do this by prefixing the file name with a percent sign ('%') as follows:

```
Command ===> %gork
```

This causes Debug to read and execute the commands contained in GORK.

Function Keys

F1: Help	provides context sensitive and field sensitive help
F2: Run	execute program: break-points are honoured, and errors are intercepted and displayed
F3: End	immediate exit back to *Go mode: the program that is currently being debugged is terminated, and all files are closed.
F4: Flip	flips display mode from INSTRUCT to MEMORY
F5: StepPast	the instruction pointed to by the PSW is executed, but the BAL and BALR instructions are not treated as branches. The effect is that all instructions in a subroutine executed using StepPast are not traced. Break-points are honoured, and errors are intercepted and displayed. Placing the cursor on an instruction in the Instruction Display causes the application program to run until it reaches that point.
F6: PrevScr	displays the previous user program screen. Pressing Enter returns to the Debug display.
F7: MemUp	previous page of memory locations is displayed
F8: MemDown	next page of memory locations is displayed
F9: Break	a break-point is installed at the indicated storage location or at the indicated address if the cursor is placed on a register; if a break-point is already set for that address it is deleted
F10: Point Data	value at the cursor position becomes the start of the display, in the Memory Display mode
F11: Point Instruction	value at the cursor position becomes the start of the display, in the Instruction Display mode
F12: Step	the instruction pointed to by the PSW is executed
PA1: Attn	When your program is running (when you are not displaying a Debug panel) your workstation enters Attention Mode. By pressing PA1 again, your application program is suspended and Debug is given control. However, the exact point at which your program will be interrupted is unpredictable.
PA1: Watch	when in Debug, this switches to the watch display. See the discussion on Watch Variables for more about this.

Debug Commands

Most Debug commands provide some sort of message. Commands are not remembered unless you prefix them with an asterisk ("*"). In this case, the command remains on the command line and can be re-issued repeatedly.

The following lists Debug commands in alphabetical order.

Attn

This command allows you to enable or disable the PA1 interruption facility of Debug. ATTN causes Debug to honour PA1 interruptions and to seize control from the application. NOATTN disables this, so that if your program is looping, you cannot re-gain control in Debug.

Break

This command allows you to specify where execution of your program should be interrupted, with control being passed to the Debug facility. You specify the address of the BEGINNING of the instruction; prior to executing this instruction you are placed into the Debug display. At this point, you can alter storage, PSW, and registers, if you wish. To resume execution, use the GO , Step , or the StepPast commands.

In the Instruction Display, the characters **Brk>** are displayed next to a breakpoint. Usage:

```
break  xxxxxx
```

where *xxxxxx* is the hexadecimal address or symbol where execution is to be interrupted, or a Map Table label.

You can set a break-point by placing the cursor at the memory location and pressing F9. As well, the value in a register can be used to set a break-point via F9. In this case, the address value in the register is used.

Calc

Hexadecimal calculations can be performed using the CALC command. Additionally, variables can be set and used in calculations. Pre-defined variables for the PSW and registers are: PSW, R0, R1, ..., R15. Expressions can be evaluated and the results displayed.

The syntax of a hex calculation is simple. An expression consists of 1 or 2 operands. The allowed operators are: +, -, *, /. A decimal value can be indicated by a number sign (#) for a decimal constant. No parentheses are allowed. You can assign either a constant, a variable, or an expression to a variable. Note that either or both operands in an expression may be variable. Examples of valid commands are:

```
calc PSW = 5000 + R15
calc R2 + #100
calc R14 = R2 + R3
calc 48F0 + CD
```

In some cases you can leave out "calc". Debug recognizes that you are requesting a calculation when you use an equal sign (=) or an arithmetic operator (+, -, *, /) in the correct position. In all of the above examples, "calc" was not necessary.

Delete

This command allows you to remove a breakpoint that you have set using the Break command. This is useful when you no longer wish control prior to executing the instruction at this address; for example, you may have stopped execution within a loop and now wish to stop after the loop has been completed. Usage:

```
delete  xxxxxx
```

where *xxxxxx* is the hexadecimal address of the breakpoint to be removed.

Display — This command allows you to view memory starting at the address that you specify in the command. Depending up the display mode that you are in the Debug display will present you with either a memory/character display, or a partial disassembly display. When you request that storage be displayed for which you are not allowed to see (typically system storage after your user region), then the message "Display address out of range" will be presented. Usage:

```
display  xxxxxx
```

where *xxxxxx* is the hexadecimal address of the beginning of storage to be displayed, or a Map Table label or symbol.

Down — This command allows you to scroll the memory display down by 4 bytes. That is, the starting address will be incremented by 4 and the display updated. This can be useful in conjunction with the * operator; placing an asterisk (*) in front of the any command causes that command to remain in the command area. In this case, it allows you to step through memory 4 bytes at a time.

Dump — This command allows you to display storage, registers, or a traceback in line-by-line mode, suitable for capturing via the /RECORD command. The "instruction" option generates a partial disassembly of storage instead of a storage dump. Usage:

```
dump regs
     traceback
     address length (instruction)
     address1-address2 (instruction)      - optional
```

End — This command causes Debug to return control immediately to *Go mode in MUSIC. This is equivalent to pressing F3 (Exit).

Entry — This command allows you to enter a named address into the Map Table. You associate a name of 8 characters or less and a hexadecimal address together, and this pair is placed in the Map Table. You can use the Locate command to search for the name, as well as the Point function key to display that address in the memory display. Once entered into the Map Table, it cannot be removed. Usage:

```
entry  nnnnnnnn xxxxxx
```

where *nnnnnnnn* is the name (max 8 characters) of the hexadecimal address *xxxxxx* to be entered into the Map Table.

Find — This command searches for character or hexadecimal strings in memory. The default search is carried out over 4096 bytes of memory, starting from the first address displayed. If you wish, you can specify a length for the search (which can be greater or shorter than 4096 bytes). Character search strings must be enclosed in quotes; for example, find 'program'. If the search parameter is not enclosed in quotes, then it is assumed to be a hexadecimal string. Usage:

```
     find  'ssssss'  length
or   find   xxxxxx   length
```

where *xxxxxx* is the hexadecimal search string, *'ssssss'* is the character search string, and *length* is an optional specification that controls how many bytes are searched

First — This command displays the first page of entries in the Map Table. You can also page up and down using the MapUp and MapDown commands, but these commands are usually used from the function keys. Use the Last command to go to the bottom of the Map Table.

Fixup — By default, if a program interruption occurs, Debug adjusts the Program Status Word to point to the instruction in error. However, the /370 hardware architecture has the next instruction address in the Program Status Word pointing after the instruction in error. In most cases, you would want Debug to perform the address fixup for you.

In case you don't wish this to be so (teaching /370 architecture, for example) issue the FIXUP command. This command toggles the address fixup for program interruptions between the default mode (Debug performing the address fixup in the PSW) and the /370 hardware architecture mode (the address in the PSW is of the instruction after the instruction in error).

Flip — This command toggles the display mode from the Memory Display (which shows the contents of main memory, the EBCDIC equivalents, and the Map Table), and the Instruction Display (which shows a partial disassembly of machine language instructions), at the current starting position. This is usually used via a function key.

The individual commands that specify screen modes are: Memory, Instruct, and Float.

Float — This command removes the Map Table and displays instead the floating point registers. These may be altered just as the general purpose registers can be, but they may not be used for addressing with Point. To restore the Map Table, use Map or Memory .

GetMap — This command retrieves the symbols from a Linkage Editor map file, and places the symbols into the Map Table. You specify the filename and optionally an offset value which will be added onto the address of each symbol entered into the Map Table. The default value is 0 if no offset is specified. Usage:

```
getmap aaaaa  nnnn
```

where 'aaaaa' is the filename to be read and 'nnnn' is the offset value. Note that you should save the map file from the linkage edit by using a /FILE statement for unit 6 (see "Linkage Editor" in Chapter 8 for more information about the Linkage Editor).

Go — This command starts executing instructions either at the current address in the Program Status Word, or at an address that you can optionally specify on the command. Instructions are executed without intervention from Debug, so that your program can "get away" from you if you have not carefully set break-points. (However, you can always re-gain control by using PA1.) All break-points are set prior to returning control to your application. Usage:

```
go    xxxxxx
```

where *xxxxxx* is an optional hexadecimal parameter indicating where execution is to commence, the default being the current instruction pointed to by the PSW

Instruct — This command switches the display mode to the Instruction Display, featuring a list of partially disassembled instructions instead of the hexadecimal/character memory display. Although you can issue this command explicitly, it is usually easier to toggle the display mode using the program function key defined for it.

IntPSW — This command allows you to specify whether the current instruction should be displayed on the screen. It works slightly differently depending upon the display mode that you are in. See the ScrUpd command as well, as this interacts with IntPSW.

In the **Memory Display** IntPSW causes the current instruction pointed to by the PSW to be partially disassembled, and placed on the line above the display of memory. NoIntPSW causes the first storage location display to be interpreted as an instruction, partially disassembled, and placed on the line above the display of memory.

In the **Instruction Display** IntPSW causes the current instruction pointed to by the PSW to be shown on the screen, when Debug initially re-gains control from the program being debugged. NoIntPSW causes the screen to remain "as-is". Usage:

```
   intpsw
or nointpsw
```

Last — This command displays the last page of entries in the Map Table. You can also page up and down using the MapUp and MapDown commands, but these commands are usually used from the function keys. Use the First command to go to the top of the Map Table.

LdLibDir — This command sets the starting address of the memory display to the in-core System Load Library directory. Note that the CREAD privilege is required. If you do not have this privilege, the message "Illegal Command" will be issued.

Locate — This command searches for a named address (symbol) in the Map Table. If found, the starting position of the Map Table is adjusted so that this name/address pair is in the top left-hand position of the Map Table display. Note that you can use Point on any symbol in the Map Table. As well, to make new entries in the Map Table use the Enter command. Usage:

```
locate   cccccccc
```

where *cccccccc* is the symbol name (max 8 characters) to be searched for in the Map Table

LPADir — This command sets the starting address of the memory display to the in-core Link Pack Area directory. Note that the CREAD privilege is required. required. If you do not have this privilege, the message "Illegal Command" will be issued.

MapDown — This command moves the Map Table display down by 1 page. This is useful to scroll down among the map entries, break-points and ENTRY labels in the Map Table display.

MapUp — This command moves the Map Table display up by 1 page. This is useful to scroll up among the map entries, break-points and ENTRY labels in the Map Table display.

MemDown — This moves the Memory Display down by 128 bytes. In other words, hexadecimal 0x80 is added onto the starting memory display address. This command is used to scroll down through main memory.

Memory — This command switches the display mode to the Memory Display, featuring a hexadecimal/character display of main storage, as well as the Map Table at the bottom of the display. Although you can issue this command explicitly, it is usually easier to toggle the display mode using the program function key defined for it.

MemUp — This moves the Memory Display up by 128 bytes. In other words, hexadecimal 0x80 is subtracted from the starting memory display address. This command is used to scroll up through main memory.

NextInstr — This command scrolls the display down by 1 instruction. This can be either 2, 4 or 6 bytes, depending upon the current instruction in the starting position of the display. This is useful when "stepping through" a program without actually executing instructions.

NucMap — This command sets the starting address of the memory display to the in-core Nucleus Map. Note that the CREAD privilege is required. If you do not required. have this privilege, the message "Illegal Command" will be issued.

PITrap — A PI (Program Interrupt) occurs due to some condition that the machine is unable to handle, such as a division by 0, an invalid operation code (bad instruction), or invalid address. PITRAP sets a mode of operation that causes MUSIC/SP to transfer control to Debug whenever a program interruption occurs. Note that with some programming languages, this may inhibit error recovery or localization, since the programming language may have its own interruption handler (eg. VS/FORTRAN).

In this case, the NOPITRAP command should be used once at start-up. This causes Debug to tell MUSIC/SP to pass control to any application interrupt handler, whether from a high level programming language or not, when a program interruption occurs. Note that the address for such a routine is usually specified via a SPIE (Specify Program Interrupt Element) or STAE (Specify Task Abnormal Exit) Assembler macro instruction. to store and keep its address there.

With judicious setting of break- points, you can debug an interrupt handler routine using Debug. Simply set a breakpoint at the beginning of the interrupt handler, and specify NOPITRAP. When your interrupt handler gains control, specify PITRAP again in order to recover from any program interruptions within the interrupt handler.

Point — This command is an extremely effective method of following address pointers to their targets. The Point command requires that the cursor be placed upon a "pointable" item, such as a general purpose register, the PSW, the Previous Address field, a word in the memory display, an address in the column of addresses, or a Map Table entry. The address at the field where the cursor is positioned is used as the starting address for the Memory display. In other words, the address is used to "point" to the next address to be displayed.

ReadBuf — This enables the refresh of the previous user program screen (by default, this is enabled). NOREADBUF disables this mode of operation. Use these commands in cases where you need to control when this function is preformed (typically, on devices where this is not supported or is slow).

REFLECT — This command affects the manner in which Debug deals with SVC (Supervisor Call) instructions. "REFLECT" refers to a performance feature in OS simulation called "fast REFLECT". This feature inhibits presenting SVCs numbered 126 and higher to the OS simulation module, as these SVCs are not standard OS SVCs but rather are MUSIC/SP SVCs. The REFLECT command sets this mode of operation.

NOREFLECT disables this mode of operation. In this operating mode, all SVCs are presented to the OS simulation module, which in turn may decide to issue a MUSIC/SP SVC. Basically, this means that all SVCs on MUSIC/SP can be seen by Debug, as opposed to only OS SVCs during OS simulation. This mode (NOREFLECT) is the default.

This command interacts with the TrapSVC command (see below).

REP — This command allows you to alter storage, starting a specified location. The syntax is:

```
REP xxxxxx sss...ss
```

where *xxxxxx* is the address in hexadecimal of where storage is to be altered, and *sss...ss* is a hexadecimal string that will be placed in storage. Examples:

```
rep 4800 0700
```

```
rep 005000 47000000
```

Both examples insert NO-OP instructions at locations 4800 and 5000 respectively.

Screen — Displays the previous user program screen, with a few restrictions. First, it will not display a Debug screen. Second, if you are single stepping through your program, actions that your program takes that alters the screen will not be preserved. This last restriction is intended to avoid congesting the channel, and to keep Debug's response time low for single step mode. If you TRACE your program, the last page full of traced instructions can be viewed. Pressing Enter returns to the Debug display. See also ReadBuf.

ScrUpd — This command allows you to control the manner in which the screen is updated. The default is that ScrUpd is false; in other words, the memory display is altered only with the Display or Point commands, in Memory Display mode. In the Instruction display, however, the screen is updated when the current instruction is no longer on the screen. (However, the IntPSW command also interacts with the ScrUpd command.)

When ScrUpd is specified, Debug calculates the first memory display position as the address in the PSW (Program Status Word). Therefore, when you are using Step or StepPast to step through your program, the current instruction is always at the top of the memory or instruction display. This can be useful, but under most circumstances, the default settings are more desirable.

Step — This command allows you to single step through an application program being debugged. Pressing the Step function key causes the instruction currently pointed to by the PSW to be executed, with control immediately returning to Debug.

In certain cases, you need to understand how Debug does this. After the instruction to be executed, Debug inserts a special SVC. When that SVC instruction is executed, Debug intercepts it and recognizes that a break-point has been encountered. This approach can cause your application program being debugged to fail strangely, in the case where it depends upon data immediately following the instruction. Note that if the instruction can cause branching, Debug puts these SVCs at both the success and failure targets of the branch.

StepPast — This command allows you to single step through an application program being debugged. Pressing the StepPast function key causes the instruction currently pointed to by the PSW to be executed, with control immediately returning to Debug. In certain cases, you need to understand how Debug does this. After the instruction to be executed, Debug inserts a special SVC. When that SVC instruction is executed, Debug gains control and recognizes that a break-point has been encountered. This approach can cause your application program being debugged to fail strangely, in the case where it depends upon data immediately following the instruction.

If you place the cursor on an instruction on the Disassembly Display panel, StepPast will execute up until that instruction. Thus, multiple instructions can be skipped, in addition to subroutine calls. This can be regarded as setting one temporary break-point. Note that control has to reach that instruction, since placing the cursor in that position does not make the program that you are debugging go there, if it would not normally go there. Thus you should be careful when subroutine calls and conditional branches are among the instructions that you are skipping.

A situation where this is useful is when you wish to pass quickly over debugged code. Another situation might be the case of a long loop which you wish to skip over; placing the cursor at the instruction after the loop and issuing the StepPast command allows you to do this.

Note that StepPast is similar to Step, except in dealing with BAL and BALR instructions. These instructions are used for subroutines calls; Debug places breakpoint SVCs only **after** the instruction (or where you place the cursor). Thus the subroutine is executed before control returns to Debug.

Top — scrolls the map display up to the 1-st entry

Trace — This command initiates "instruction tracing", which basically displays a message about each instruction as it executes on the workstation. This is useful in the case where you wish to see the instructions that are being performed, but don't wish to use Step or StepPast due to the number of instructions to be executed. An optional parameter allows you to specify how many instructions are to be traced; the default is 20 instructions (a screen-full). If you specify 0, then there is no limit to instruction tracing and it will halt only on a PI (Program Interrupt) or when control transfers to protected storage (and therefore a break-point cannot be set!).

An useful approach is to trace instructions with /RECORD NEW or ON. This provides you with the opportunity to display the instruction trace and compare it with the program that generated it. Usage:

```
trace  nnnn
```

where *nnnn* is the number of instructions to trace. Use the TRACE parameter of the Debug command to suppress the display of traced instructions on the display screen.

TrapSVC — This command intercepts SVCs (Supervisor Call instructions), and causes Debug to display the current debugging screen with the message 'SVC Intercepted'. It functions only when SVC trapping is operating (normally when OS simulation is active).

There are basically two forms of SVC trapping: selective and complete. Selective trapping allows you to indicate that specific SVCs only are to be trapped. Complete trapping intercepts almost all SVCs. Note that the REFLECT/NOREFLECT command interacts with this command in that it controls which SVCs can be seen by user region programs such as Debug.

The TrapSVC command accepts various parameters. They are

trapsvc nn — causes hexadecimal SVC number "nn" to be intercepted. In the case where "nn" might not be interpreted as a hexadecimal number, prefix it with a leading zero (eg. "0nn"). Note that you can specify as many SVCs as you wish to be trapped; all of these selections will remain valid until you reset them. This command implies "trapsvc on" and "trapsvc select".

trapsvc all — causes all SVCs to be trapped. Note that "trapsvc on" is implied. See also "trapsvc select".

trapsvc on — causes SVC trapping to be turned on. This enables SVC trapping for **all** SVCs. This implies "trapsvc all".

trapsvc off — causes SVC trapping to be turned off. Any SVCs that were previously selected remain selected, although they will not be trapped (your list of SVCs to be trapped is still valid). To re-enable trapping of these SVCs, specify "trapsvc on". To completely reset trapping for all SVCs, specify "trapsvc reset". The start-up state can be restored by specifying: reset, all, off.

trapsvc select — causes SVC trapping to be performed only for previously specified SVC

numbers. You can switch between trapping all SVCs (trapsvc all) and trapping only SVCs that you have individually specified (trapsvc select). Note that this implies "trapsvc on".

trapsvc reset — causes the list of previously selected SVCs to be cleared. Note that this does not affect whether SVC trapping is enabled or note. To turn SVC trapping off, specify "trapsvc off".

Up — This command allows you to scroll the memory display up by 4 bytes. That is, the starting address will be decremented by 4 and the display updated. This can be useful in conjunction with the * operator; placing an asterisk (*) in front of any command causes that command to remain in the command area. In this case, it allows you to step up through memory 4 bytes at a time.

VIP — This command informs Debug that it can take advantage of the user's VIP privilege. Note that this privilege must be acquired prior to starting Debug. However, Debug will not allow stores into protected storage until you specifically tell Debug to do so via this command. This protects you from accidentally altering sensitive areas in storage without realizing it.

Normally, you enable VIP, perform your alterations, and disable VIP again by re-issuing the VIP command. Debug tells you whether you are enabling or disabling VIP mode explicitly, after you issue the command.

Watch — This command switches the display to the WatchPoint panel. See the discussion on watchpoints for more information.

WatchPoint Facility

A "watchpoint" is an area in storage that is monitored, even when your program is running. When it is altered, your program is interrupted. Debug has the ability to monitor watchpoints and to interrupt execution when one changes. You access the Display WatchPoints screen (Figure 10.4) via PA1 when in Debug's instruction or data displays. You can change the watched storage, in one of 3 formats: CHAR, INT, and HEX. You can add, delete, and toggle the watchpoints between active and inactive. You provide any name for the watchpoint; Debug uses this name when it reports a change to the watchpoint.

Whenever Debug gains control, it automatically checks the watchpoints and informs you of the first watchpoint to have changed. Thus, Debug checks this on breakpoints, SVCs, and traced instruction boundaries. Execution of your program is suspended if it was in progress.

In order to check on a watchpoint, Debug must be in control. The process of being passed control and checking the watchpoint may require several hundred machine language instructions (at least). This can have a significant effect on the execution speed of your program.

If you elect to check watchpoints after every instruction, then you will be slowing down your program by a factor of several hundred or so. However, you will identify the exact instruction that changed the watchpoint.

If you elect to check watchpoints after ever branch instruction, given that branches occur on average every 5 to 10 machine instructions, your program may run 5 to 10 times faster than the preceding case. The disadvantage, of course, is that you cannot identify precisely where the change to the watchpoint occurred (but it will be within a few instructions).

Checking watchpoints at every SVC has essentially no impact on execution speed. However, you may find that this is of little help in pinpointing where the watchpoint changed. Keeping in mind that tens of

thousands of instructions may be executed between SVC instructions, you can see that this option is sometimes useless.

The following screen is displayed when you press PA1 from the main Debug display:

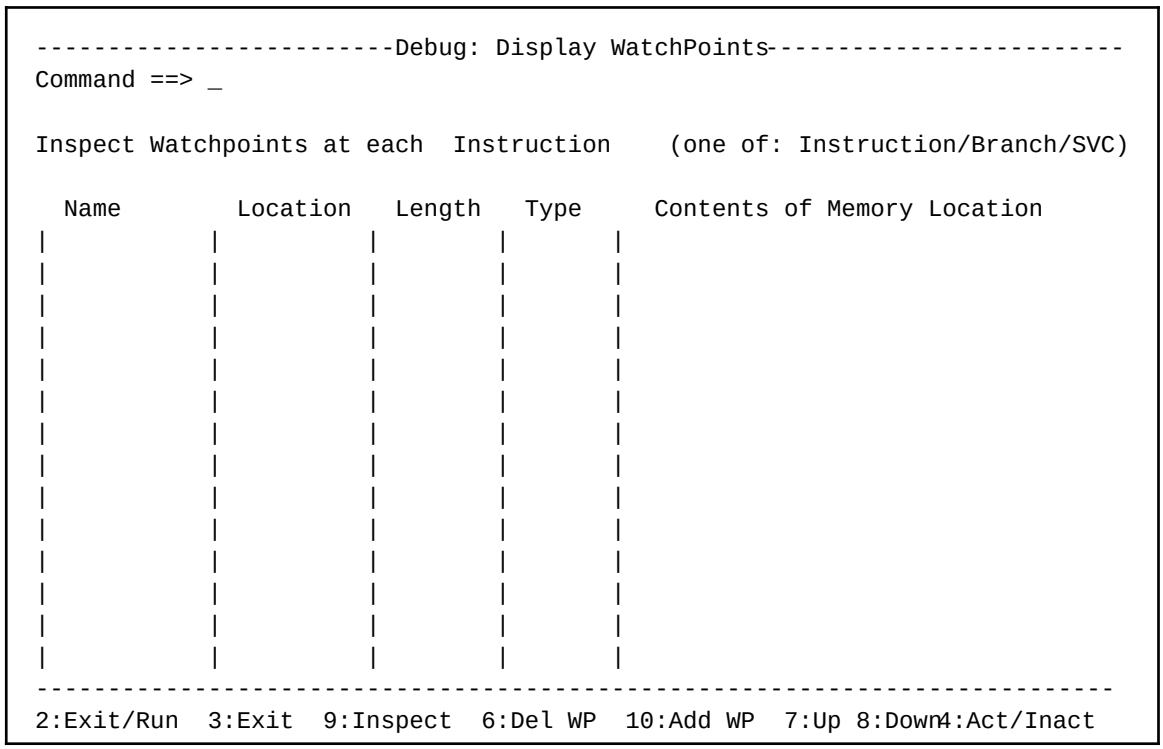


Figure 10.4 - Displaying Watchpoints

On the Display WatchPoints screen (figure 10.5) you can view and change the information that is displayed. Up to 200 watchpoints can be added.

Function Keys

F2: Exit/Run executes your instructions immediately.

F3: Exit returns to the main Debug display.

F9: Inspect instructs Debug when to check watchpoints; at each instruction, branch, or SVC. In terms of CPU and elapsed time, the Instruction and Branch choices slow down your program significantly.

F6: Del WP deletes the watchpoint that your cursor points to.

F10: Add WP goes to the Add Watchpoint panel for adding new watchpoints.

F7: Up moves up by one screen.

F8: Down moves down by one screen.

F4: Act/Inact leaves a watchpoint definition there, but renders it inactive. To re-activate it press F4 again. Active watchpoints are highlighted on the screen, while inactive ones are not. When there are no active watchpoints, Debug does not stop your program to check on them; and so

execution speed is not affected by inactive watchpoints.

Adding a WatchPoint

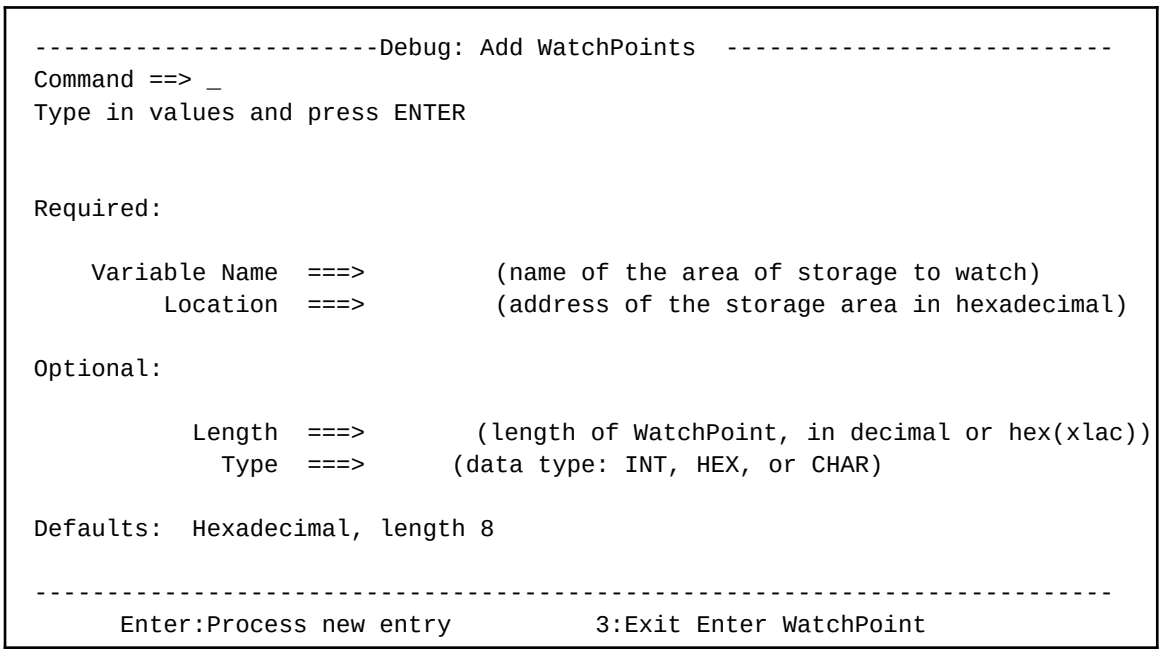


Figure 10.5 - Adding Watchpoints

To add a watchpoint, enter an arbitrary name (Debug uses this to report the status of the watchpoint) and an address. Length and type are optional; the default is hexadecimal type of length 8. The length may be entered in hexadecimal; in this case, preface the hex length with an x (i.e. x1AC). Press ENTER after you have typed all of your specifications.

Repeat this for as many watchpoints as you require. Then press F3 to exit this panel, and return to the Display Watchpoint panel (Figure 10.4), where you should see the watchpoints that you have just entered. By default, they are set to "active" status.

Copying one UDS File To Another - DSCOPY

The DSCOPY utility program copies from one User Data Set (UDS) file to another. Both UDS files should have the same record size (RSIZ). Use the UTIL program to do the copy if the record sizes are not the same.

The input data set is defined on unit 1 and should be specified as SHR. The output data set is defined on unit 2 and must be specified as OLD or NEW.

The program is invoked by:

```
/FILE 1 UDS(fromdsname) VOL(volume) SHR
/FILE 2 UDS(todsname) VOL(volume) OLD
/INCLUDE DSCOPY
```

The copy operation is done rapidly by copying multiple blocks of the file at a time, rather than record by record. Normally the two data sets are equal in size. If the receiving data set is smaller, a warning is issued and as many blocks as possible are copied. (A block is 512 bytes.)

DSLIST

The DSLIST program is used to produce an alphabetical list of all the User Data Set (UDS) files that you currently own. The listing can be displayed on your workstation or it can be written on unit 10 for subsequent saving as a file. (You can use the LIBRARY command to get a listing of the files that you currently own.)

The program is invoked by:

```
DSLIST
```

or through a file like the following, called SAMPLE

```
/INCLUDE DSLIST
OUTPUT=n
```

where *n* is the output unit number. (For example, OUTPUT=10.)

Examples

```
dslist
*In progress

DATA SETS FOR CODE ABCD     31JUL73      3 DATA SETS    40 TRACKS

DSNAME      VOLUME     LRECL TRKS   NREC  CREATED LASTREF

ABCDFIL1    MUSIC1       80    20   2400  25JUL73 31AUG73
ABCDFIL2    MUSIC2       80    10   1200  01JUL73 15DEC74 BACKUP
ABCDLMOD    MUSIC2      128    10    800  31JUL73 07APR75

*End
*Go
/input
/inc dslist
output=10
/endrun
*In progress

DATA SETS FOR CODE ABCD     31JUL73      3 DATA SETS    40 TRACKS
*End
*Go
save mylist,sv
*In progress
SAVED,     3 RECORDS
*End
*Go
list mylist
*In progress
ABCDFIL1    MUSIC1       80    20   2400  25JUL73 31AUG73
ABCDFIL2    MUSIC2       80    10   1200  01JUL73 15DEC74 BACKUP
```

```
ABCDLMOD    MUSIC2      128    10    800  31JUL73 07APR75
*End
*Go
```

DSREN

UDS Rename Program

The DSREN program allows users to change the name of their UDS files. This rename function for UDS files is similar to the RENAME command that applies to your files. For system security reasons, you must run this program from a workstation and NOT from batch. Additionally, you can only rename files that belong to you.

The contents of the UDS file is not altered by the rename function. The new name that you assign to your file must conform to the usual UDS naming conventions discussed under the /FILE statement. The rename operation will be rejected if one of your files already exists with that name and is located on the same disk volume.

Using DSREN

The rename program is invoked by simply typing DSREN when your workstation is in command (*Go) mode. (Some installations may require you to type in the full command form of /EXEC DSREN.)

The DSREN facility will ask you to specify the disk volume name, and the old and new data set names. These three items must be enclosed in single quotation marks as in the example shown below:

```
VOL='MUSIC1',OLDNAM='dsname',NEWNAM='dsname'
```

After the rename operation is completed, you may enter information for another rename operation. When you are finished all the rename operations simply enter a blank line or type /CANCEL.

Non-Conversational DSREN

Normally you run the DSREN program conversationally, that is, the rename information is typed in when the program is running. You may also use this facility with a previously prepared list of rename operations as shown below:

```
/INCLUDE DSREN
VOL='MUSIC1'.... etc
VOL='MUSIC1'.... etc
.......
```

The above control statements may be saved in a file for more convenient usage at a later time.

You may also use a INPUT=n option to direct the rename program to read the rename data from MUSIC I/O unit number n. This INPUT specification may be entered at the time you would normally enter the rename information. This option is useful when the rename data is to be read from a UDS file. An appropriate /FILE statement must appear before the /INCLUDE DSREN control statement if the input is being taken from a UDS file.

ENCRYPT/DECRYPT

ENCRYPT and DECRYPT are two programs that respectively encrypt (code) and decrypt (de-code) files to provide added security. When you encrypt your files you are prompted for a 1 to 8 character password. To decipher the encrypted data you must use the identical password. You must therefore keep careful track of which password is associated with each file.

The ENCRYPT program randomly exchanges characters resulting in an unintelligible document. To restore your file to its original form, use the DECRYPT program along with the encryption password for that file. If you forget or lose the password for an encrypted file, you have essentially lost the contents of that file.

The files can have a fixed or variable record format fixed and can be compressed or uncompressed. The record length of the file can be from 1 to 32760 bytes.

Parameters

infile — is the input file to be encrypted.

outfile — is the output file to which the encrypted data is written.

PW(password) — is the encryption/decryption key used to perform the respective function. The same password must be used for ENCRYPT and DECRYPT.

REPLACE(xx) — where *xx* is ON or OFF. When REPLACE is on, the output file is replaced without verification, if it exists. If no output file is specified then the input is replaced. The default is REPLACE (OFF).

Note: For the replace keyword any abbreviation is acceptable, for example all of these are correct:

r(on), re(on) repl(on), ... replace(on) etc...

Examples

1. In this example WORK is the input file and WORK1 is the encrypted output file.

```
ENCRYPT work work1
```

   After entering the above, you are prompted for a password, and if WORK1 already exists you are asked if it should be replaced.

2. This example deciphers the data in WORK1 and creates WORK2. You will be prompted for the encryption password, but will not be asked if you wish to replace WORK2 should it exist.

```
DECRYPT work1 work2 REPL(ON)
```

3. In this example an output file name is not provided and the password is added. In this case, you are not prompted for a password, but you will be asked if you wish to replace the input file WORK.

```
ENCRYPT work PW(tpsecret)
```

Return Codes

In order to assist you in using ENCRYPT and DECRYPT from programs such as REXX the following codes are returned upon termination of the respective program.

0= normal return
1= error in opening input file or RECFM is U
2= error invalid parameter specification.
3= error in the encryption/decryption routine.

EXARCH and EXREST

EXARCH and EXREST are two programs that can be used to copy all or some of your MUSIC files to and from tape. Both programs must specify RECFM(U) on the /FILE statement. Use of these programs is primarily for exporting MUSIC files to non-MUSIC installations. They are NOT substitutes for FILARC and FILRST, which are used for archiving and restoring files for use solely on MUSIC installations.

EXARCH

To copy your MUSIC files to tape use EXARCH.

General Form

```
/FILE 1 TAPE VOL(vvvvvv) RECFM(U)
/INCLUDE EXARCH
options
filename1
filename2
   .
   .
   .
```

vvvvvv is the volume name of your tape (up to 6 characters).

options list of options, separated by comma:

CODE='userid' where *userid* is your ownership id (userid without subcode), in quotes. Specify this option only if you wish to archive all the files belonging to your userid.

HEADER=T to have header records at the start of each file on the tape. T is the default.

HEADER=F suppresses the header records.

If HEADER=T, then a header record containing the file name, logical record length, and length of file is written. This is needed in order to run EXREST; EXREST reads the information on this record, enabling it to restore the desired files.

filename1,filename2,etc...
are the filenames of your MUSIC files, one file name per line. Used when not all files are to be archived.

Examples:

1. This example copies your entire library (all files) to tape with header records.

```
/FILE 1 TAPE VOL(MYTAPE) RECFM(U)
/INCLUDE EXARCH
CODE='ABCD',HEADER=T
```

2. This example copies files FRED, TESTPROG and TRY99 to tape without header records.

```
/FILE 1 TAPE VOL(MYTAPE) RECFM(U)
/INCLUDE EXARCH
HEADER=F
FRED
TESTPROG
TRY99
```

Notes:

1. Each run of EXARCH writes over any previous files on the tape.

2. Each file saved is a separate file on the tape.

3. If EXARCH is successful the files will be copied in the order they are listed.

4. The difference between EXARCH and FILARC is that the latter does not produce tapes compatible with non-MUSIC installations, and each file is in a compressed form on the tape.

5. This procedure does not delete your files from MUSIC.

6. After running EXARCH, a printout of what is on the tape is produced. It is important to attach this to the tape when transporting it to another installation as it contains vital information (blocksize, logical record length and record format) needed to read it. All tapes are created as non-labelled. Normal data files are dumped in standard MVS type tape formats (VB or FBS). Details of these formats can be found in OS/VS2 MVS Data Management Services Guide, IBM Form #GC26-3875.

7. If a MUSIC load module (created by the Linkage Editor) is to be dumped by EXARCH, it should be created with RECFM(U) or RECFM(F), and for RECFM(F) the record length must be a divisor of 512, for example LRECL(128). LRECL(80) must not be used.

EXREST

In order to copy your MUSIC files from tape use EXREST. EXREST can only be used for files if copied on to tape through EXARCH, and if HEADER was not specified as F.

General Form

```
/FILE 1 TAPE VOL(vvvvvv) RECFM(U) SHR
/INCLUDE EXREST
NAME='name1',TO='name2',options
NAME='name1',TO='name2'
```

vvvvvv is the volume name of your tape (up to 6 characters).

name1 is the name of the file on the tape.

name2 is the desired MUSIC file name.

name1 can be the same as name2, and can take on the following forms:

```
'name'
'userid:name'
'userid:prefix*'
'prefix*'
```

If the last character is an asterisk (*), all files whose names start with the specified prefix will be restored. If the plus sign is used, *name1* must have the "*" as well as *name2*, and all files with such a prefix will be restored. I.e.:

```
NAME='ABCD:TT*',TO='ABCD:TEMP*'
```

This restores all files of the form ABCD:TTxxx to ABCD:TEMPxxx, where xxx is any character string.

options — list of options, separated by commas, specified on the first parameter statement only:

- ALL=T — to copy all files from the tape. The files would be restored with the same file names they have on the tape.
- ALL=F — is used when the *name* parameter is used. This is the default.
- Repl=T — to replace existing files.
- Repl=F — is specified if files are not to be replaced. This is the default.
- FIXUP='x' — FIXUP is used for identifying FILES restored whose names already existed as MUSIC files. *x* is the character for fixup. The default fixup character is "$". This is useful when REPL=F yet you want the file to be restored. Identification of the file will be by the file name with the FIXUP character added to the end. To disable the fixup character, use FIXUP=' '. If the fixup character was disabled, REPL=F was specified, and the file already existed, then the file to be restored will not be copied into a MUSIC file, and the old MUSIC file will be the only one that exists.
- FILES=n1,n2,... — n1,n2,... are the tape file sequence numbers of those files you wish to restore (in addition to those specified by name). The maximum number of file numbers which can be specified is 200.

Examples:

1. This example restores all your tape files to MUSIC files, replacing any MUSIC files that have the same name as those on the tape.

```
/FILE 1 TAPE VOL(MYTAPE) RECFM(U) SHR
/INCLUDE EXREST
ALL=T,REPL=T
```

2. This example restores files with the prefix T, changing the prefix to AT, and restoring one file (FRED) having the same name that was on the tape. If any of the file names exist they are not replaced, but rather the character $ is added to the end of the name of the file being restored.

```
/FILE 1 TAPE VOL(MYTAPE) RECFM(U) SHR
/INCLUDE EXREST
NAME='T*',TO='AT*',FIXUP='$'
NAME='FRED'
```

3. This example restores a tape file (FRED) under a different name (WORK) than what was on the tape.

```
/FILE 1 TAPE VOL(MYTAPE) RECFM(U) SHR
/INCLUDE EXREST
NAME='FRED',TO='WORK'
```

FILARC/FILRST/FILCHK

Dumping and Restoring MUSIC Files

The programs FILARC and FILRST may be used to archive (dump) a group of files to cards or tape and later restore one or more of the dumped files. The FILARC program copies files to cards or tape, preserving all attributes of the files such as record length, record format, and tag field. The original files are NOT deleted by the archive program. The FILRST program restores specified files to the user's library, using as input the dump produced by FILARC. FILRST can rename files as they are restored. A third program, FILCHK, reads and checks a dump produced by FILARC. FILCHK is useful for ensuring that a good dump exists before purging the files, and also for finding what files are on an archive tape.

The archive is in the form of card images (record length 80), but files of any record length may be archived and restored. The data is dumped in a special format which only the MUSIC system can use. For this reason, FILARC must not be used for transporting files to non-MUSIC systems. A dump produced by FILARC is usable only by the FILRST program. If you wish to transport files to a non-MUSIC installation, use the UTIL program to dump them to tape in sequential, logical record form.

Control Statements

The sequential file containing the archive dump is defined by a /FILE statement for unit 1. The logical record length of this file is 80. Tape, a UDS, or a file may be used. The control statements shown below use tape.

The /FILE statement is followed by a /INCLUDE for the desired program (FILARC, FILRST, or FILCHK), then by a parameter statement which specifies the program options to be used. Options on the parameter statements must be separated by commas, and are of the form KEYWORD=VALUE, where VALUE is a number, or a character string enclosed in single quotes, or T (for true), or F (for false). Most options have a default value, which is the value used by the program if the option does not appear on a parameter statement.

File Archive (FILARC)

```
/FILE 1 TAPE VOL(vvvvvv) LRECL(80) BLK(nnnnn)
/INCLUDE FILARC
parameters separated by commas (see below)
filename DUMPNAME=dname   )
filename DUMPNAME=dname   )   optional additional file names
...                       )
```

CODE='userid' This specifies that all files of this userid are to be archived. *userid* must be the ownership id (userid without subcode) of the user running the program. If the UDATE parameter is used, only files satisfying the date requirement are archived. A list of additional files to be archived may follow the parameter statement. Each file name must start in column 1, and may be of the form *userid:name* or *name.* If *userid:* is omitted from a file name, the user's userid is assumed. If you only want to archive some of your files, omit the CODE parameter and supply a list of the file names.

Normally, a file is copied onto tape using the name it has on the system. If this is not

desired, then the DUMPNAME parameter can be given to specify another name. In any case, the file's name is not changed on the system.

UDATE='ddmonyy'
: This specifies a date in the form: 2-digit day, 3-character month abbreviation, and 2-digit year (for example 08FEB89), to be used in conjunction with the CODE parameter. Only files whose last-read and last-written dates are both less than or equal to UDATE are archived. This gives a means of archiving old files. The parameter CODE='userid' should be specified when UDATE is used. The UDATE parameter does not apply to any specific file names specified.

NAMES=n
: This requests the names of all archived files to be written to unit number *n*. This is useful if you wish to later purge the files. If this parameter is omitted, the names are only displayed.

TAPFIL=n
: When archiving to tape, this parameter specifies the file sequence number to be written to on the tape. The default is TAPFIL=1. Note that writing to file *n* does not disturb existing files 1 through *n-1*, but destroys any following files on the tape.

File Restore (FILRST)

```
/FILE 1 TAPE VOL(vvvvvv) LRECL(80) BLK(nnnnn) SHR
/INCLUDE FILRST
first parameter statement (see parameters below)
second parameter statement
...
```

Each parameter statement gives the name of a file (or group of files) to be searched for in the archive dump, and optionally gives the name of the file (or group of files) to which the file (or group) is to be restored. The NAME and TO parameters are used for this. The first parameter statement has the first NAME/TO combination and also any other options to be used during the entire restore job (TAPFIL, REPL, etc.). The second and following parameter statements specify additional NAME/TO combinations. See below for examples.

NAME='userid:name' or NAME='userid:prefix*'
: This specifies the full file name (including user code) to be searched for in the archive. If the last character is an asterisk (*), all files whose names start with the specified prefix will be restored. NAME= may be abbreviated N=. Up to 2000 NAME parameters can be specified per job.

TO='userid:name' or TO='userid:prefix*'
: The full file name (including user code) to which the file is to be restored. The user code must be specified, and must match the code of the user running the program. If the TO parameter is omitted, the original file name will be used. If the last character is an asterisk (*), then the NAME parameter must also have +, and the indicated name changes are made. For example, NAME='ABCD:TT*',TO='ABCD:TEMP.*' restores all files of the form ABCD:TTxxx to ABCD:TEMP.xxx, where xxx is any character string. Resulting names are truncated if too long. TO= may be abbreviated T=.

TAPFIL=n
: The tape file sequence number containing the archive dump to be used. The default is TAPFIL=1. This parameter is needed only when reading from a multi-file tape.

REPL=T or REPL=F
: REPL=T causes the "TO" file to be replaced if it already exists. If REPL=F is used, the existing file is not replaced and the restore program may try to use a different name (see FIXUP below). The default is REPL=F.

FIXUP='x' This specifies a character x to be used for generating an alternate file name when the "TO" file already exists (and should not be replaced), or cannot be written to. The fixup character is added to the end of the filename, or replaces the last character if the name is already the maximum length. At most three tries are made. The default fixup character is the dollar sign ($). To prevent fixup attempts, specify a blank, as in FIXUP=' '.

Dump Checkout (FILCHK)

```
/FILE 1 TAPE VOL(vvvvvv) LRECL(80) BLK(nnnnn) SHR
/INCLUDE FILCHK
parameters separated by commas (see below)
```

NAMES=T or NAMES=F

Use the NAMES=T parameter if you want a listing of all the file names contained on the archive tape. The default is NAMES=F.

INFO=T or INFO=F

Use the INFO=T parameter to print the information line for each file contained on the archive tape. The information line has the file name, logical record length, record format (e.g. FC for Fixed Compressed), access control options, total space allocated (units of 1k = 1024 bytes), the number of 512-byte blocks dumped, and the time and date the file was archived. The default is INFO=F.

TAPFIL=n Use the TAPFIL=n parameter to specify that tape file *n* is to be checked for the archive output. The default is TAPFIL=1, meaning the first file on the tape is to be checked. This parameter is used only when reading from a multiple file tape.

Examples:

1. This example archives three files to tape. The files are assumed to be under the userid of the user running the program.

```
/FILE 1 TAPE VOL(ARCTAP) LRECL(80) BLK(4000) OLD
/INCLUDE FILARC
        <--- (blank line, no parameters)
PROG1.SOURCE
PROG1.OBJECT
DATA.TEST1
```

2. This example is a batch job to archive all files for the userid ABCD which have not been referenced since March 31, 1988. The names of these files are written to file ARC.NAMES.

```
/FILE 1 TAPE VOL(MYTAPE) LRECL(80) BLK(4000) OLD
/FILE 2 NAME(ARC.NAMES) NEW
/INCLUDE FILARC
CODE='ABCD',UDATE='31MAR88',NAMES=2
```

3. This example archives two files to punch cards. The files must be public files, or MM15 must be the userid of the person running the batch job. Note that the punched cards would have to be copied to tape or to a UDS or file before they could be used by FILRST.

```
/FILE 1 PUN
/INCLUDE FILARC
        <--- (blank line, no parameters)
MM15:DOC1
MM15:DOC2
```

4. This example reads and verifies an archive tape. The file names will be printed.

```
/FILE 1 TAPE VOL(MYTAPE) LRECL(80) BLK(4000) SHR
/INCLUDE FILCHK
NAMES=T
```

5. This example restores all files for userid ABCD, and renames the files by putting the characters "OLD." at the beginning of the names. Any existing file ABCD:OLD.xxx will not be replaced, since REPL=F is specified.

```
/FILE 1 TAPE VOL(MYTAPE) LRECL(80) BLK(4000) SHR
/INCLUDE FILRST
REPL=F,NAME='ABCD:*',TO='ABCD:OLD.*'
```

6. This example restores four files from an archive tape. File PROG6 will be restored under the new name PROG6X. The other files will be restored under their original names, and any existing files will be replaced (REPL=T).

```
/FILE 1 TAPE VOL(MYTAPE) LRECL(80) BLK(4000) SHR
/INCLUDE FILRST
N='ABCD:FILE35',REPL=T
N='ABCD:DATA.FILE'
N='ABCD:PROG6',T='ABCD:PROG6X'
N='ABCD:FILE13'
```

FTP and TELNET from MUSIC

FTP and TELNET are now available from MUSIC. TELNET allows you to establish sessions with computers on the TCP/IP Internet. FTP allows you to transfer files. Many computers on the network maintain libraries of public files that you can access through FTP. These files include public domain software, graphics and games. Check with your installation to see if this facility is available at your site.

Connection

Each computer in the network is assigned a name which is used on the FTP, GOPHER, or TELNET command to establish a connection. For example the command

```
TELNET VM1.MCGILL.CA
```

will create a session on the VM1 system at McGill. The command

```
FTP VM1.MCGILL.CA
```

will create a file transfer connection.

The NET command is useful in locating the names of computers connected on the Internet. This displays a list of computers that offer various services such as FTP access to public files. You can browse through the list and locate specific items. An FTP, GOPHER, or TELNET connection can be established directly from NET by typing an F, G, or T in the margin beside the particular entry.

There are two MUSIC commands that can help you check the connection between you and the remote site. They are:

FINGER — The FINGER command allows you to send a one-line query to a remote Internet site, and receive back information on who is logged into that remote site.

PING — The PING (Packet Internet Groper) command measures round-trip-times to internet sites. It does this by sending an ICMP/IP echo request to the remote site, which IP is obliged to respond to if it receives the packet.

See *Chapter 5 - MUSIC Commands* for details about FINGER and PING.

Login

Once you are connected to the remote computer you usually have to login by entering a userid and a password. Many FTP sites accept the special userid of "anonymous", allowing you to connect and access their public file library without having a userid assigned on their system. You can usually enter anything for the password.

Documentation

MUSIC uses the VM TCP/IP product to provide FTP and TELNET services. Complete FTP and TELNET documentation are in the book *IBM Transmission Control Protocol/Internet Protocol for VM* (GC09-1204).

For complete details about FTP and TELNET refer to the *MUSIC/SP Communications Guide*.

Printing Save Libraries

The LBLIST program can be used to display the contents of all the user's files that exist on the Save Library. (Execute-only files and files with record format U (undefined format) will, however, not be displayed.) Files with record length longer than 100 bytes will be truncated to 100 bytes on the printout. This program is normally run from batch, due to the large amount of output that will usually be produced. The following gives the control statement set up:

```
/INCLUDE LBLIST
...Parameter statement (see below)
```

Parameter Statement:

```
CODE='xxxx'[,NONEWPAG][,ALLOBJ]
```

CODE='xxxx' This parameter specifies the 1-16 character ownership id (userid without subcode). It must agree with the ownership id on the /ID statement.

NONEWPAG This parameter is used to suppress page skips between files. If this parameter is omitted, then each file will start at the top of a new page.

ALLOBJ This parameter causes all object module statements to be printed. If this parameter is omitted, then TXT and RLD statements will not be printed. Since TXT and RLD statements contain, for the most part, unreadable information, it is unlikely that you will want to print them.

Fullscreen Panel Generator

Using the IBM 3270 type workstation, it is possible to display an entire screen full of information and have the user enter multiple data items with only a single I/O operation. This is referred to as full-screen I/O. The PANEL system has been written to allow easy access to full-screen I/O from high-level languages. All that is needed is some understanding of the basic concepts of screen formatting. No knowledge of 3270 protocol is required.

A formatted screen image, hereafter called a *panel*, is composed of fields. A field may contain, for example, explanatory text or a prompting message or it may be used to accept input data from the user. Fields may be placed anywhere on the screen, although they may not overlap and they must start and end on a single line. A single line on the screen may contain multiple fields.

Each field is immediately preceded by one character that is used to define the attributes of the field. This attribute character controls various features of the field which are discussed later. Attribute characters are invisible; the fact that they must be present means that there must always be at least a one-character gap between fields.

A field may be displayed in normal or in intensified display (highlighting), or it may be hidden. Hidden fields are normally used for entering sensitive information, such as passwords. A field may be either modifiable or protected from modification by the workstation user. If the user tries to enter data in a protected field, the workstation will reject the input by locking out the keyboard.

An application program must use panels which have been previously designed and stored, and can make use of as many different panels as it needs. A panel is essentially a subroutine stored as an object deck in a file. Only one panel per file is allowed. When the subroutine is called by an application program, it will display the defined fields on the user's screen in their proper positions and with the desired characteristics. Data passed from the calling program will be mapped into the proper fields on the screen.

After the panel has been displayed, the user can enter data into any unprotected field on the screen. When the user presses an *action key* such as ENTER or a Program Function key, then the data entered can be passed back to the calling program.

Services Provided by PANEL

The following is a summary of the services performed automatically by the PANEL utility.

- Basic full-screen I/O
  Any call to a panel normally results in screen I/O. The screen image and the data supplied by the calling program are written to the workstation. Data subsequently entered by the user will be passed back to the calling program.

- Action key translation
  When user input is returned to the application program, a two byte code is also returned indicating which action key the user pressed. See the discussion entitled "Accessing Panels" for a complete list of the codes.

- Cursor Positioning

When a panel is displayed, the application program may specify where on the screen the cursor should initially be positioned. If this positioning is not specified explicitly, then a default position will be used.

After screen I/O is complete, the application program receives a code indicating where on the screen the cursor was positioned when the action key was pressed.

- Audible alarm
  Some 3270s have an audible alarm (beeper) built in. An application program has the option of sounding this alarm when a panel is written to the screen. This can be useful in drawing the user's attention to the fact that the user has made a mistake.

- Translation of input to upper case
  Like most workstations, 3270s are capable of transmitting both lower and upper case letters. Most users enter all their data in lower case, while most programs use the corresponding upper case text. The PANEL system will normally convert user input to upper case, but the option can be disabled for any panel.

- Function Keys 13-24
  Not all 3270 keyboards have 24 Program Function keys. Since most applications have no need for as many as 24 function keys anyway, it is usually convenient to define the keys 13 through 24 as being equivalent to 1 through 12. An option is available to cause the keys 13-24 to be translated as though the corresponding key 1-12 had been pressed. Thus the application program would never be aware that a key 13-24 was used.

- Automatic panel printing
  Sometimes it is useful to be able to get a copy of a screen image on paper, or in a file. An option exists which allows a user to print the panel using the F12 key. The panel is written to a file with ddname FMTPRINT.

- Automatic HELP facility
  An excellent way of helping users to use an online system is to build in a HELP function. PANEL's HELP facility allows a HELP panel, or a series of HELP panels, to be associated with any given panel. A HELP panel is simply another panel, presumably containing explanatory text on the use of the application.

Developing Panels

To invoke the PANEL program, simply type PANEL when in Command (*Go) mode. You will be requested to enter the name of a file into which the panel subroutine is to be saved (the OUTPUT FILE), and optionally, the name of an INPUT FILE. You can also type the file name as a parameter on the PANEL command, for example, "PANEL mypanfile".

If you are creating a new panel, the OUTPUT FILE should be a new file, or if it exists, it should be empty.

If you are modifying an existing panel the OUTPUT FILE should be the name of the file containing the existing panel. At the end of the modification process, the modified panel will overwrite the old panel in the same file.

Alternately, if you are modifying an existing panel, you may specify the file containing the existing panel as the INPUT FILE and specify a different file for the OUTPUT FILE. At the end of the modification process, the modified panel will be saved in the OUTPUT FILE with the original panel unchanged in the INPUT FILE. This is a useful technique for producing a number of similar panels from a single *skeleton* panel.

To display a panel file without changing it, use the command "SHOPAN filename" (or "CHOPIN filename").

Each accessible field is numbered and filled with a ruler.

Creating a New Panel

Once the initial file information has been correctly entered, a blank screen will be displayed. (This will not occur if you are modifying an existing panel; instead refer below to "Modifying an Existing Panel".)

Two kinds of fields can be created: accessible fields and text fields. Accessible fields are those which are used for communication between the calling program and the 3270 screen. An accessible field may be protected or unprotected from user modification. A protected accessible field is usually used by the program to send messages to the user; an unprotected accessible field usually receives information from the user that is to be passed back to the calling program. (See the description of the PROTECT and UNPROTECT keys under "Modifying an Existing Panel" below.) Text fields, on the other hand, are displayed on the screen, but are never modified by the user or the calling program. Titles, headings and the like would normally be text fields.

To specify an accessible field in your format, type in underscores (___) in the appropriate place on the screen. The length of the field is determined by the number of underscores entered.

To specify a text field, simply type the text in the desired place on the screen. A text field may not contain underscores.

Note that, as stated previously, at least one space must precede each field. This space is used by PANEL to insert the attribute character for the field. Note also that no field may be split between two screen lines.

Returning now to the blank screen that PANEL has presented, you are now ready to place the text and accessible fields on the screen. Use the cursor positioning keys (right, left, up, and down) to move the cursor to the start of a field. *DO NOT* press the ENTER key or any function key until you have completed the first draft of the panel. Pressing these keys takes you out of the creation phase and into the panel modification phase.

While entering the various fields of the new panel, should you wish to correct mistakes or change the order of field appearance, simply blank out the mistake and retype the field. Once the initial layout is complete, press the ENTER key. The layout will be processed as follows:

- all text fields will be protected
- all accessible fields will be high-lighted and left unprotected

The panel will then be echoed back to your screen and you may then proceed to modify it.

Modifying an Existing Panel

You are now ready to modify the panel in order to finalize its appearance and characteristics. If you were modifying an existing panel, rather than creating a new one, you would have arrived immediately at this stage, bypassing the initial creation stage.

Modifications to panels are accomplished by using Program Function keys. The keys PF1-PF12 have designated functions as listed below. The keys PF13-PF24, found on some workstations, have meanings identical to the keys 1 through 12. That is, PF13 and PF1, PF14 and PF2, PF24 and PF12, are equivalent.

For most functions, the cursor position at the time the function key is pressed is crucial. For instance, if F9 (PROTECT FIELD) is pressed, then the field at which the cursor is positioned is the one which will become

protected.

Modification Function Keys

Function keys for use in modifying panels:

1/13 HELP & set F2	2/14 HIDE or CURSOR or NOSKIP	3/15 END
4/16 CREATE or SPLIT field	5/17 DELETE or TRUNCATE field	6/18 Flip Field INTENSITY
7/19 Move Fields UP	8/20 Move Fields DOWN	9/21 PROTECT Field
10/22 Move Field LEFT	11/23 Move Field RIGHT	12/24 UNPROTECT Field

Figure 10.6 - Function Keys for PANEL

F1 HELP — This key will cause a HELP panel to be displayed. The HELP panel gives a short summary of the functions of each of the modification function keys. It also allows you to change the function of F2 from HIDE (the default) to CURSOR or to NOSKIP. Pressing F3 will return you to the original panel.

F2 — The F2 key can have one of three possible functions: HIDE (the default), CURSOR or NOSKIP. To change the function of this key, simply display the HELP panel by pressing F1, and type the desired function into the F2 square.

F2 HIDE — A field on the screen may be *hidden* (invisible). Although any kind of field (i.e. text or accessible, protected or unprotected) may be hidden, the only field it would normally make sense to hide is an accessible, unprotected field. This attribute is usually reserved for entering sensitive data such as a password.

F2 CURSOR — When a program calls upon a panel to be displayed, it may specify where the cursor is to be positioned, or it may allow the cursor to be positioned by default. You can indicate which field is to be the default by moving the cursor to the start of a field and pressing the F2 key. If you do not use the CURSOR function, the default will be the first unprotected field on the panel.

F2 NOSKIP — When a user types data into the last character of a field, the cursor normally will skip to the start of the next input field. To prevent this skipping, you can set *noskip* on any input field by moving the cursor to that field and pressing the F2 key. When a

user types data into the last character of a *noskip* field, the cursor jumps 2 characters to the right.

F3 END — When you are satisfied with the appearance of the panel, press F3 to end modification and proceed to the termination stage. If you have made any changes to the panel you should press ENTER before pressing F3. Pressing ENTER causes PANEL to echo the screen ensuring that all changes have been correctly stored.

F4 CREATE — This key may be used either to create a new field, or to split an old one in two. In either case, the position immediately to the left of the cursor must be a space, or the operation will fail. The space is required in order that an attribute byte may be inserted for the new field.

If the CREATE key is pressed when the cursor is not positioned at an existing field, a new field will be created. It will be an unprotected, normal intensity field and will extend to the end of the screen line, or up to the start of the next field if one exists on the same line.

If the CREATE key is pressed when the cursor is at an existing field, that field will be split into two, i.e., an attribute byte will be inserted at the space to the left of the cursor. (Remember that a space must exist to the left of the cursor when you press the CREATE key.) The display intensity of the field to the right of the split will be opposite of the display intensity to the left of the split (as though F6 had been pressed), but other attributes will remain the same.

F5 DELETE — This key is used to either truncate a field or to delete the entire field depending on where the cursor is when F5 is pressed. If the cursor is at the first byte of the field, then the entire field will be deleted. If the cursor is somewhere in the middle of the field, then the field will be truncated at that point. The cursor position marks the first character to be truncated, NOT the last one to be kept.

If you have just created a text field, make sure you press PROTECT before you press TRUNCATE.

Note that truncation of a field MUST be done using the DELETE key. Blanking out underscores in an accessible field does not have any effect on the actual length of the field.

F6 INTENSITY — The Flip Field INTENSITY function changes the display intensity of a field:

- a normal-display field will be highlighted
- a highlighted field will become normal
- a hidden (invisible) field will become normal

F7 UP — The UP key moves all the lines, from the line where the cursor is positioned down to the bottom of the screen, up one line. This is accomplished by deleting the entire line immediately above the cursor. The line above the cursor is deleted even if it contains a field. While this is a convenient way of deleting unwanted fields, beware of inadvertent destruction.

F8 DOWN — The DOWN key moves all the lines, from the cursor down, down one line, and inserts a blank line ABOVE the cursor. The operation will fail if the last line on the screen (line 24) is not empty.

F9 PROTECT — This key will protect a field - that is, make it impossible for the workstation user to type over it. This is normally used to define a text field, although you may protect an accessible field. Protecting an accessible field makes it available to the calling

program while ensuring that the user cannot modify its contents. This is normally done for fields used by the program to present error messages to the user.

The correct use of this key is important when a new text field is to be created. To do this, you must first CREATE an accessible field, then type the desired text into that field. Then, ensuring that the cursor is still positioned somewhere in the field, press the PROTECT key. The field will become a protected text field, and only then may be truncated, high-lighted, etc. The PROTECT key must be used IMMEDIATELY after the text has been typed into the field.

F10 LEFT — This is used to shift a field one column to the left. Note that the action applies only to a single field, not to a group of fields. There must be room available to move the field to the left or the operation will fail.

F11 RIGHT — This is used to shift a field one column to the right. Similar rules apply to RIGHT as to LEFT.

F12 UNPROTECT — Used to make a field accessible and unprotected. All unprotected fields must be accessible fields, so any text in a field which you unprotect will be discarded and replaced by underscores. Information that the user types into an unprotected field on a panel is passed back to the calling program.

Examples of Modifying a Panel

PANEL requires a specific sequence of actions for creating new fields, or modifying existing ones:

- To CREATE a new TEXT field:

  Move the cursor to the desired position of the start of the field and press the CREATE key (F4).

  Type out the text.

  Press the PROTECT key (F9).

  Press the TRUNCATE key (F5).

- To CREATE a new ACCESSIBLE field:

  Move the cursor to the desired position of the start of the field and press the CREATE key (F4).

  Type out the length - (using digits 1 through 0 makes it easier to count out the desired length.)

  Press the TRUNCATE key (F5).

- To change a TEXT field into an ACCESSIBLE field:

  If you are completely replacing the field then move the cursor to the field and press the UNPROTECT key (F12).

  If you are splitting a field, then move the cursor one character to the left of where the new field is to begin (leaving a space for the attribute character) and carry out the following steps:

  Press the TRUNCATE key (F5).

Move the cursor 1 character to the right; otherwise the next action will be ignored.

Press the CREATE key (F4).

Type out the length as above.

Press the TRUNCATE key (F5).

Storing Panels

When the END key (F3) is pressed to terminate panel modification, you will be presented with a termination panel. On this you should specify

1. The name that is to be given to the panel, i.e., the name of the *subroutine* that an application program will call to have the panel displayed.

2. The options to be used when displaying the panel such as translating user input to uppercase, translating function keys 13 through 24 into function keys 1 to 12, using PA2 as automatic panel reshow, or using F12 to automatically print the panel in a file having a data definition name (ddname) of FMTPRINT.

3. The HELP panel subroutine name to be associated with this panel (if any). The specified HELP panel will be linked to the present panel and be available to the user if the user needs assistance while viewing the present panel. If this is a HELP panel you are storing, you can enter the subroutine name of the next HELP panel (if any). Each HELP panel is developed and stored separately, but one HELP panel may be linked to another in this manner, thus allowing an unlimited number of HELP panels to be displayed when a user needs assistance. The user can scroll through these HELP panels using F1, F10 and F11.

   When the user presses the F1 key, the first HELP panel associated with the user's current panel will be displayed. Using the F11 or ENTER keys (show next HELP panel) and the F10 key (show previous HELP panel), the user can scroll through the entire series of HELP panels associated with the original panel. Pressing F3 will get the user out of help mode and back to the original panel. Figure 10.7 shows this flow in a diagrammatic form.

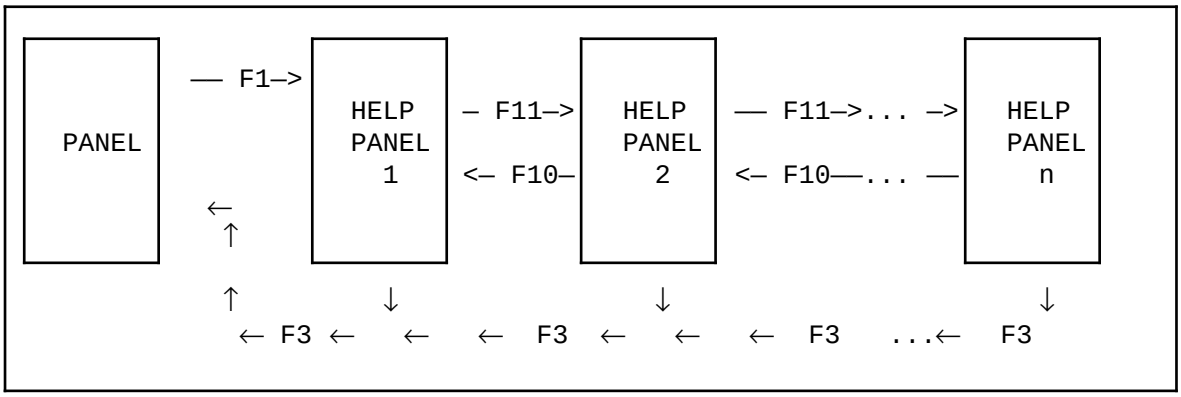


Figure 10.7 - Flow of Panels

Once you have filled in the options, press ENTER to complete the creation of the panel object deck. You will now be returned to the initial screen, from which you may commence creating or modifying another panel. To exit PANEL, press the F3 (END) key.

Accessing Panels through a Program

Once you have created your panel, it can be called from an application program exactly the same way as any other subroutine. Two arguments may be supplied, although only the first one is required.

Calling Sequence: CALL panel-name(io-area,supplemental)

where *panel-name* is the subroutine name given to a panel when it was stored.

The IO-AREA Parameter

The first parameter is called the I/O area. It contains some basic control information plus the data to be passed between the calling program and the panel. It is a block of data whose length in bytes must be no less than the total length of all accessible fields in the panel, plus 6.

For example, suppose there are four accessible fields of lengths 5, 5, 10 and 24. To call this panel, an I/O area no less than 50 (5 + 5 + 10 + 24 + 6) bytes long must be passed. The I/O area is laid out as follows:

Bytes	Function
1-2	Attention ID (AID) This two character code represents the action key which the user pressed after the data was entered. It is set by the panel display service before control is returned to the application program. The codes for the various action keys are as follows: '01' - F1 'EN' - Enter '02' - F2 'A1' - PA1 . 'CL' - Clear . 'CS' - Cursor Select '24' - F24 *Notes:* 1. If a HELP panel is available then '01' will never be returned. 2. If the automatic print panel was selected then '12' will never be returned. 3. If automatic translation of function keys was selected then codes '13' through '24' will never be returned.
3-6	Cursor Position These four bytes contain two integer halfwords declared in Fortran as INTEGER*2, in COBOL as PICTURE S99 COMPUTATIONAL-1, or in PL/1 as FIXED BINARY(15,0). These two halfwords can be used to determine where the cursor will be positioned when a panel is first displayed. When control is passed back to the calling program, these halfwords indicate where the cursor was positioned when the action key was pressed. Depending on the contents of the SUPPLEMENTAL parameter (discussed below) the cursor position can be given by either the panel field number or the panel row and column number. Panel Field Number: When specifying cursor position by panel field number, bytes 3 and 4 of IO-AREA should

contain the accessible field number on the panel; bytes 5 and 6 should contain zero.

Row and Column Number
When specifying cursor position by row and column number, bytes 3 and 4 of IO-AREA should contain the row number (1-24); bytes 5 and 6 should contain the column number (1-80).

7 to end — Accessible Fields
The remainder of the IO-AREA is in one-to-one correspondence with the accessible fields of the panel. Each accessible field, starting with the first one and proceeding left to right, top to bottom through the panel, has a number of bytes equal to its own length reserved for it in this portion of the IO-AREA. It is through these fields that information filled in by the user can be passed back to the calling program.

The SUPPLEMENTAL Parameter

This second argument may optionally be passed to a panel subroutine. It contains *supplementary orders* which give access to the alarm and cursor override services.

The orders are contained in a 16-byte field, of which currently only the first 4 bytes are meaningful. The layout of the orders is as follows.

Bytes	Function
1	Panel Function Normally, this byte is set to a blank. The following codes are available for this field: P If the Function field is set to P, then the panel is written to the screen, but *no read from the workstation is done*. This is useful for display only panels - control is returned to the program immediately after the panel is written to the screen. G If the Function field is set to G, then a read is performed *without* displaying the panel. A Code A will do a write followed by an asynchronous read. The program regains control immediately after the write is complete. When the user presses an action key the data is read and the application woken up with the POST-CODE set to ATTN. The application can retrieve the data using the R (Read only) request. If the application issues a subsequent PANEL request (other than R), the asynchronous read is cancelled. R Code R reads data from asynchronous read request. *Note:* Setting the Function field to any value other than a blank, P, G, A, or R will probably cause your program to abend.
2	Alarm. Set this to 'A' to sound the 3270's alarm when the panel is displayed. This can be useful in drawing the user's attention if the user has made a mistake.
3	Cursor Position as Field Number To explicitly specify cursor positioning as using a field number move "C" to this position, and set the cursor position code in the IO-AREA to the desired field number. "L" means leave cursor alone.
4	Cursor Position as Row and Column To specify cursor position as using row and column numbers, place an 'X' in this position and put the row number in the first half-word and the column number in the second half-word of the cursor position in the IO-AREA parameter.
5-16	These positions should be left blank.

Modifying Panels Under Program Control

Subroutines MODOPT and MODFLD can be used in your program to dynamically modify panel options and data field attributes. See MODOPT and MODFLD in *Chapter 11. System Subroutines*.

Coding examples

Example of a panel:

```
                SAMPLE PANEL

     ENTER CUSTOMER NUMBER ==> _____
           PURCHASE ORDER ===> _____
           INVOICE AMOUNT ===> __________


     MESSAGE: ________________________
```

Figure 10.8 - Sample Panel

Note that, in the *sample* panel above, the underscore characters are accessible fields which will appear when modifying the panel, but will not be present when the panel is displayed under program control.

COBOL example:

```
01  FMT-IO-AREA.
    02  FMT-AID          PIC XX.
    02  FMT-CURSOR.
        03  FMT-CRS-FLD  PIC S99  COMP.
        03  FILLER       PIC S99  COMP.
    02  FMT-CUST-NO      PIC X(5).
    02  FMT-PUR-ORDER    PIC X(5).
    02  FMT-INV-AMOUNT   PIC X(10).
    02  FMT-MESSAGE      PIC X(24).
          . . .
PROCESS.
    MOVE SPACES TO FMT-CUST-NO
                   FMT-PUR-ORDER
                   FMT-INV-AMOUNT.
    MOVE 'FILL IN AND PRESS ENTER' TO FMT-MESSAGE.
    CALL 'SAMPLE' USING FMT-IO-AREA.
* FINISH PROCESSING WHEN USER PRESSES F3
    IF FMT-AID = '03'
        GO TO PROCESS-EXIT.
    MOVE FMT-CUST-NO TO OUTPUT-CUST-NO.
    MOVE FMT-PUR-ORDER TO OUTPUT-PUR-ORDER.
    MOVE FMT-INV-AMOUNT TO OUTPUT-INVOICE-AMOUNT.
    WRITE OUT-REC FROM OUTPUT-RECORD.
    GO TO PROCESS.
          . . .
```

FORTRAN example:

```
      INTEGER*2 AID,CURSOR(2),ENTER/'EN'/,PF3/'03'/
      LOGICAL*1 CUSTNO(5),PURCH(5),AMOUNT(10),MESSAG(24)
            . . .
      COMMON /IOAREA/ AID,CURSOR,CUSTNO,PURCH,AMOUNT,MESSAG
C
C  the common block groups the variables together into a
C  "structure" so that all the variables are contiguous and
C  in the required order. Thus passing the first variable in the
C  common block results in the whole structure being passed.
            . . .
   10 CALL FILL(CUSTNO,44,' ')
      CALL LMOVE('FILL IN AND PRESS ENTER',MESSAG,23)
      CALL SAMPLE(AID)
C  FINISH PROCESSING WHEN USER PRESSES F3
      IF (AID.EQ.PF3) GO TO 999
      WRITE(1,100) CUSTNO,PURCH,AMOUNT
      GO TO 10
           . . .
```

PL/I example:

```
 DCL IO_AREA   CHAR(50),
   1 IO_STRUCTURE DEFINED(IO_AREA),
   2 AID       CHAR(2),
   2 ROW       FIXED BIN(15,0),
   2 COLUMN    FIXED BIN(15,0),
   2 DATA,
   3   CUSTOMER_NUMBER   CHAR(5),
   3   PURCHASE_ORDER    CHAR(5),
   3   INVOICE_AMOUNT    CHAR(10),
   2 MESSAGE   CHAR(24);
DCL SAMPLE ENTRY(CHAR(50)) OPTIONS(ASSEMBLER INTER);
      .  .  .
  CUSTOMER_NUMBER, PURCHASE_ORDER, INVOICE_AMOUNT = ' ';
  MESSAGE = 'FILL IN AND PRESS ENTER';
  DO WHILE(AID •= '03');
     CALL SAMPLE(IO_AREA);
     SELECT(AID);
        WHEN('03');
        WHEN('EN') DO;
          PUT FILE(OUT) LIST(CUSTOMER_NUMBER,
            PURCHASE_ORDER,INVOICE_AMOUNT);
  CUSTOMER_NUMBER, PURCHASE_ORDER, INVOICE_AMOUNT = ' ';
  MESSAGE = 'FILL IN AND PRESS ENTER';
         END;
        OTHERWISE MESSAGE = 'HIT ENTER TO ENTER DATA';
     END;
    .  .  .
```

Usage

After you have created your panels, you must ensure that they and the Panel Display Service subroutines (stored in the file PANEL.SERVICE) are accessible to the linkage editor when compiling your application program, just as you do for any other subroutines.

Consider an example of a FORTRAN application program using a panel stored as a subroutine called SAMPLE in the file called SAMPLE.OBJ:

```
*Go
list sample.obj
*In progress
/LOAD VSFORT
      . . .
      CALL SAMPLE(AID)
      . . .
      END
/INCLUDE SAMPLE.OBJ
/INCLUDE PANEL.SERVICE
*End
*Go
```

Panel Printing

If the automatic print option was specified when a panel was stored (see the topic "Storing Panels" above) a user can have a copy of the displayed panel printed to a file. The calling program must have a /FILE statement specifying a file with a ddname of FMTPRINT, for example:

```
/FILE FMTPRINT NAME(...) ...
```

If these requirements are meet, the displayed panel will be printed to the specified file when the user presses the F12 key. When this is done, control is not returned to the calling program, but rather a message appears on the screen indicating that the panel has been printed, and the panel is redisplayed.

REXX interface to PANEL

An interface to MUSIC's PANEL facility is available through the special PANEL command in REXX. The screens are created as usual via the PANEL facility and stored as object files. When REXX encounters a PANEL command it dynamically loads the appropriate object files for the screen images. Modifiable fields are then filled from the specified REXX variables and the panel service routine is invoked to perform the I/O. When this is complete, the data from any modified fields is then returned via the REXX variables.

The PANEL facility is invoked from REXX using the PANEL command. Due to the fact that the interface modifies items in the variable list, the entire PANEL command should be enclosed in quotes.

```
'PANEL  filename   [ var-list ]'
```

filename The name of the file containing the object file for the panel to use.

var-list A list of variable names separated by blanks indicating which REXX variable is associated with a specific field on the screen.

Field Variables:

1. The fields on the screen are numbered from left to right, top to bottom.

2. The first name in the list is assigned to the first field, the second name to the second field, etc.

3. There need not be a variable name specified for each field on the screen or in fact for any fields on the screen. In either case, the default variable name of *name.n* is used, where *n* is the number of the field and *name* is the panel file name, with leading ownership id and directory names (if any) removed. For example, for file name "ABC:TEST\MYPAN", the default variable names are MYPAN.1, MYPAN.2, etc.

4. A period (".") can be used in the variable list to force the system to use the default name.

5. When a field is initialized from a variable, nulls are appended to the end if the variable is not long enough to fill the field.

6. On returning from PANEL trailing blanks and nulls are removed.

7. The panel's subroutine name is ignored by REXX.

Special Variables:

The following variables are used in addition to the panel fields themselves. These names cannot be overridden by the user.

AID Two character code representing the action key pressed by the user.

FIELD Numeric value used to position the cursor to a specific field. It is also returned to indicate the position of the cursor when the action key was struck. The fields are numbered 1, 2, 3...

ROW Row number used as an alternative to FIELD.

COL Column number used as an alternative to FIELD

PANCTL Four character supplemental parameter used to control the operation of the alarm and determine whether field or row/column positioning is used.

RC This variable is set to the return code as follows.

100 - insufficient storage to load panel
101 - PANEL command is invalid
102 - file is not a panel object file
103 - panel file too large
104 - error accessing REXX variable
105 - error converting ROW, COL, or FIELD to numeric
<100 - file system return code from reading panel file

For further details on the values that these variables can have refer to the documentation on the PANEL-Subsystem.

Help Panels

There are two names associated with a panel: the name of the file that contains it, and the subroutine name that is used by languages such as FORTRAN to invoke it. The link between a panel and it's help panels is normally provided through the subroutine name. Since REXX dynamically loads the panels into memory based on the file name, the file name and the subroutine name MUST be the same for help panels.

Example

```
/INC REXX
   /* This is an example of a data entry program.
      Fields from a panel are written to a file.
      The panel is called ENTDAT and the file DATFIL. */

   do forever

     /* Display the panel and get the data. Stop if
        F3 is pressed */

      'PANEL ENTDAT NAME ADDR.1 ADDR.2 ADDR.3 PHONE '
      if rc•=0 then do;say ' Error in ENTDAT panel'; exit; end;
      if aid=3 then leave

     /* write the data to the file */

      queue name || addr.1 || addr.2 || addr.3 || phone
      MUSIO WRITE DATFIL 1
    end
```

Note: The PANEL command line must be in upper case characters.

POLYSOLVE

Overview

The POLYSOLVE program is a powerful conversational tool that can be used as a *desk calculator* facility, demonstration program, and it can even be used to solve equations! A sample POLYSOLVE session is given at the end of this writeup.

One powerful feature of this program, (the one that it takes its name from), is its ability to solve polynomial equations. The following are examples of polynomial equations:

```
x=15/40
x + 64 = 0
17x^2 + 47x = 16
16= x^3 + 5x
x= sqrt(15)
ax^3 + bx^2 +cx +d = 0
```

You will notice that you cannot solve the last equation immediately because of the unknown coefficients of a,b,c and d. The POLYSOLVE program will realize that and ask you to define them. This will be discussed later. Some more common every-day uses of polysolve are shown in the examples below:

```
15 + 63 + 27 + 14/3 + 2.49
sqrt(15.3)+63/2.3
24*3                          (The "*" means multiplication)
```

Usage Notes

The program will solve any polynomial in X with a maximum of 25 real coefficients. All computations are carried out in double precision. First order polynomials are solved algebraically. Higher order polynomials are solved iteratively, by the double precision form of the SSP routine POLRT.

The program will solve polynomials for the unknown x entered according to the following conventions:

1. An expression not involving X is assumed to mean X=expression, and the value of X is evaluated and displayed. For example, if the expression 2+2 is entered, the result 4 is displayed.

2. An expression in X without an equal sign is assumed to mean expression=0 and the value or values of X are evaluated and displayed.

3. An expression in X written with an equal sign is taken as is.

All coefficients must be real numbers. For example, SQRT(-2) is invalid.

Constants

Constants are stored in double precision form. The maximum value of a constant is approximately 10**50.

Variables

The independent variable must be X. Coefficients may be represented by a single letter from A to Z except X. Use of E should be avoided because of possible confusion with exponential notation for constants. Variables are stored as double precision numbers. The highest power of X which may be entered is 20.

The following operators are recognized:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation (Raising to a power. **2 means squared)
'	exponentiation (equivalent to **)

Implied Operations

Parentheses imply multiplication if no explicit operator is given. For example 3(A+B) is taken as 3 times the sum of A and B.

A constant immediately followed by a variable or a function name is assumed to have a multiplication operator in between. For example 3X is taken as 3 times X, and 3SQRT(B) is taken as 3 times the square root of B. However 3XSQRT(B) is invalid since it would be taken as 3 times XSQRT(B) and there is no function called XSQRT. X followed by a constant is assumed to have an exponentiation symbol in between. For example X2 is taken as X to the power 2.

Blanks are always ignored.

Functions

The following functions are supported:

ALOG	log to base e
EXP	e to a power
SQRT	square root
SIN	sine of an angle expressed in radians
COS	cosine of an angle expressed in radians

The double precision forms of these functions are automatically used.

Entering a Polynomial

1. An expression not involving X written without an equal sign. For example 3A*SQRT(B) is taken as X=3A*SQRT(B).

2. An expression in X written without an equal sign. For example 3X2+X3-3 is taken as 3X2+X3=0.

3. An expression in X written with an equal sign. For example 3X2+SQRT(B)=-2.

Expressions may not:

1. Begin with an equal sign.

2. Be an expression not involving X written with an equal sign. For example A=B+C is invalid

3. Contain an X or an = within parentheses.

Continuing an Expression

An expression is assumed to be continued on the next line if it ends in a comma, or is completely blank, or ends with the right parentheses count lower than the left parentheses count, or the last character is an operator or an equal sign (=).

A function name must not be split between lines, nor can a constant be split between lines.

Entering Coefficients

If variable coefficients are used in the polynomial, the system will request values for these variables. These values may be entered as constants or as expressions involving other variables.

The values may be entered explicitly in the form variable = constants or expressions in the order in which they are requested. Implicit and explicit entries may be intermixed on one line. Expressions must be preceded by an equal sign.

If more than one value is entered on a line, they must be separated by commas.

The continuation rules shown above under "Entering A Polynomial" also apply for entering coefficients.

Solution

The value or values of X are displayed by the program. Complex roots are shown as real part and imaginary part. The letter "i" follows the imaginary part in the printout.

Example

```
*Go
polysolve
*In progress

Enter calculation or type HELP or /CAN
?
3(x+1)=3
Incorrect character sequence "(X+1)" found near column 4

?
3x+3=3
Ans
        .0

Enter calculation or type HELP or /CAN
?
sqrt(-1)
IHN261I DSQRT NEGATIVE ARGUMENT=-0.9999999999999997D+00
Solution terminated

Enter calculation or type HELP or /CAN
?
1403+75-155/158+3*2501+
Continue statement
?
145.5
Ans
        9125.52

Enter calculation or type HELP or /CAN
?
6sqrt(b)=3x

Enter B.
?
=p+3f

Enter P,F.
?
f=c-d,25.5

Enter C,D.
?
d=1,10
Ans
        14.4914

Solve old equation again?
Type yes or no
?
no

Enter calculation or type HELP or /CAN
```

```
?
20.5x3 + 10x2 +32.3x=15/2.5 + sqrt(2.3)
Ans
        .212616
       -.350210      -1.26566   i
       -.350210       1.26566   i

Enter calculation or type HELP or /CAN
?
/cancel
*Terminated
*Go
```

PROFILE

Overview

Your user profile contains such things as your sign-on password, default tab character and tab columns, job time limits, etc. The PROFILE program is used to change or display these options.

To use the PROFILE program, enter the command "PROFILE". You will be asked to enter options. These options are listed below, in alphabetical order. If you like, you can enter more than one of these options on the same line, separated by blanks or commas. Also, if only one command line is to be entered, you can type it on the PROFILE command line, for example: PROFILE SHOW. Entering a blank command is equivalent to SHOW.

Note: Any changes made to your profile options (except passwords) do not take effect until the next time you sign on.

Options

ALWAYSPROG(x) Gives the file name of a program that is to be executed whenever your workstation would otherwise be in *Go mode. For example, this could be a menu program that lets you choose the next thing to do. You will be prompted to enter your sign-on password. To undefine the always program, type ALWAYSPROG(). You are not allowed to change the name if it is marked as FIXED. Abbreviations: ALWAYS, ALPROG

Note: If the always program does not work correctly (or does not exist), and you get stuck in a loop, don't panic! Go into attention mode (by pressing PA1 or BREAK), then type /CAN ALL. This should return you to *Go mode. You can then run PROFILE and change the always program name.

AUTOPROG(x) Gives the file name of a program that will automatically be executed each time you sign on. The name can be up to 22 characters long. To remove the name, type AUTOPROG(). You are not allowed to change the autoprog name if it is marked as not cancellable. Abbreviations: AUTO, PROG

BACKSPACE(x) Defines *x* (any special character) as the input backspace character (not necessary on 3270-type workstations). The character may also be entered as BACKSPACE(xx), where *xx* is the 2-digit hexadecimal form. To undefine the backspace character, type BACKSPACE(). See also TERMINAL(x). Abbreviations: BACKSP, BS

BATCHPW(x) Defines a new batch password (1 to 8 characters). This password may be required when you run a batch job. To set no batch password, type BATCHPW(). Abbreviation: BPW

BRIEF Displays your name and id number (if assigned). Abbreviation: BR

DEFTIME(n) Sets the default time limit for workstation jobs, in service units (SU). To specify no limit, type DEFTIME(NOLIM). Abbreviations: DEFT, DEF, DT

EDMSG Sets the conversational read prompt to `*Enter data` rather than the normal "?".

END — Terminates the PROFILE program. Must be entered at the beginning of the command line.

EDNFERR — This option applies to the /EDIT command, and means that if the file specified on the command does not exist, the Editor should give an error message and stop.

EDNFOK — This option applies to the /EDIT command, and means that if the file specified on the command does not exist, the Editor should start with an empty file of that name.

Note: EDNFERR and EDNFOK should not both be on; if they are, EDNFOK has priority. If neither is on, the result depends on how the Editor was configured for your site.

EXEC — The *implied* form of the EXEC command is not allowed. To execute a file, you must type the full command "EXEC filename", rather than just "filename".

HELP — Displays a description of the PROFILE program and options. Must be entered at the beginning of the command line. The text displayed is taken from the public file PROFILE.HELP.

HEX(x y) — Defines an input replacement character. When the system sees *x* in an input terminal line, it will replace it with *y*. *x* may be any special character (other than "/", comma, or blank), and *y* may be any character. Each may be given in 2-digit hexadecimal form. They are separated by a blank or comma. For example: HEX(&,02). To remove this option, type HEX().

INPROG — Removes the NOINPROG option.

INVIS — Makes your id invisible to programs like FINGER. You can use this option if you do not wish other users to know that you are signed on.

ITABS(n1 n2 n3 ...) — Defines n1, n2, n3, ... as the default input tab column positions. Each number is a column position from 1 to 80. When you enter a tab character, the system will supply the appropriate number of blanks to get to the next tab position. See the TAB option. To undefine all input tab columns, type ITABS(). The numbers are separated by blanks or commas. Abbreviation: ITAB

LANGUAGE(national language name) — This specifies the national language you prefer for messages, etc. The change will take effect the next time you sign on. Not all applications support all languages. If an application does not support the language you request, it uses English. National language names are: English, French, Kanji (Japanese), Portuguese, and Spanish. Other language names MAY also be accepted. Most language names have a 2 or 3-character abbreviation, for example LANG(ENG). To remove the LANGUAGE option, specify LANGUAGE() or LANGUAGE(DEFAULT), both of which use the site-dependent default language. Abbreviation: LANG

NEWPW — If you wish to be prompted to enter the new password (1 to 8 characters) in a blacked-out area (or non-display area for 3270-type workstations), use this option instead of PASSWORD(x).

NOEDMSG — Removes the EDMSG option.

NOEXEC — Removes the EXEC option. Allows the *implied* EXEC command to be used.

NOINPROG — Suppresses the `*In progress` message.

NOINVIS	Removes the INVIS option.
NOSLASH	Removes the SLASH option. "/" may be omitted on commands in *Go mode.
OTABS(n1 n2 n3 ...)	Defines n1, n2, n3, ... as the default output tab column positions. The numbers must each be 1 to 254 and be in increasing order. They are separated by blanks or commas. Up to 11 numbers can be specified. On some types of workstations, the system can use output tabs to speed up output. The user must set the correct physical output tabs at the workstation. See also the TERMINAL(x) option. To undefine the output tabs, type OTABS(). Abbreviation: OTAB
PASSWORD(x)	Specifies a new sign-on password (1 to 8 characters). You will be prompted to enter your old password. See also NEWPW. Abbreviations: PASSW, PW
PRINT	Same as SHOW
PRINT$	Same as SHOW$
PRINTSPACE	Same as SHOWSPACE
ROUTE(name)	Assigns the default route destination for your userid.
SHOW	Displays your complete profile. A blank command line is equivalent to SHOW.
SHOW$	Displays the dollar limit for your userid and the amount used so far. These values are also included in the output of the SHOW command. Abbreviation: $
SHOWSPACE	Displays the current total space occupied by your files (in units of K = 1024 bytes), and also your file space limits (if any). Abbreviation: SHOWSP
SLASH	A slash character ("/") is required on all command in *Go mode.
TAB(x)	Defines *x* (any special character) as your logical input tab character. To specify the character in 2-digit hexadecimal form, type TAB(xx). To undefine it, type TAB(). See the ITABS option.
TERMINAL(x)	Defines the type of workstation you use. 3270, 3101, and PCWS are examples of workstation type names. A workstation name must be defined in order for the OTABS (output tabs) and BACKSPACE options to be honoured. Refer to the trmcls parameter of the /ID command in *Chapter 5. MUSIC Commands* for more details. To remove this option, type TERMINAL(). Abbreviation: TERM

Example

```
*Go
profile
*In progress

USER PROFILE - ENTER COMMAND OR HELP
?
noinprog itabs(10 20 30)
CHANGED
?
show
```

```
  USERID=SPQR            FILE OWNERSHIP ID=SPQR       TYPE=0
  ID=987-6543          NAME=Julius Caesar
  TIME LIMITS (IN SERVICE UNITS):
    PRIME=180  NONPRIME=180  BATCH=180  DEFAULT=60
  MAX NUMBER OF EXTRA SESSIONS PER TERMINAL:     5
  PASSWORD CAN BE CHANGED BY USER
  SNGL: CONCURRENT SESSIONS (ON DIFFERENT TERMINALS) ARE NOT ALLOWED
  NOCOM: FILES MAY BE SAVED PRIV OR SHR, BUT NOT PUBL
  AUTOPROG:  (NONE)
  INPUT TABS:    10  20  30
  NO OUTPUT TABS
  FUNDS ($):      257.35 USED,       500.00 LIMIT
  SAVE LIBRARY:  TOTAL =    132K   LIMIT =    200K   MAX/FILE =   400
  MAX TRACKS PER DATA SET (UDS) AT ALLOCATION:        0
  CREATED 1991/03/15  (YEAR/MONTH/DAY)
  LAST SIGN-ON: 1991/10/31 10:20   LAST BATCH JOB: 0
  LAST PASSWORD CHANGE:   SIGN-ON PW 1991/10/31   BATCH PW 1991/10/31
  PASSWORD LIFETIME:  NO LIMIT
  USERID OF CREATOR:  $000000
?
end
*End
*Go
```

Sorting

Overview of MUSIC SORT Facilities

MUSIC provides facilities for users to sort fixed-length files in main storage and on disk. Disk sort operations can handle large numbers of individual records. MUSIC's disk sorting capability features the fact that it preserves the input sequence for those records that contain identical information in the sort fields.

The following identifies the various sorting facilities available on MUSIC. The disk sort routines are then fully described individually in the same order as they are highlighted below.

SORT	This command, valid during *Go mode, sorts the contents of a file. Refer to *Chapter 5. MUSIC Commands*.
MNSORT	Generalized disk sort program allowing you to sort files without the need for writing any program.
SSORT	Routine to sort arrays or tables in main storage. The sorted array replaces the original array. No disk work utility files are used, nor is any work storage array required. This routine is suitable for the smaller sorting requirements. This subroutine is callable from many of MUSIC's high-level languages such as FORTRAN. Input sequence may not be preserved for those records that contain identical information in the sort fields. For more details about this routine, refer to the SSORT subroutine writeup in *Chapter 9. System Subroutines* in this guide.
DSORT	Generalized disk sort subroutine callable from many of MUSIC's high-level languages such as FORTRAN.
SRTMUS	Subroutine similar to DSORT, except that the parameters are specified in a format similar to the OS Sort/Merge routine available on other operating systems.
COBOL SORT	The MUSIC disk sort facility is directly callable from COBOL using the COBOL SORT Verb.
PL/I SORT	A PL/I program can perform sorting operations by calling any of the sort interface subroutines PLISRTA, PLISRTB, PLISRTC and PLISRTD, as described in the PL/I Optimizing Compiler Programmer's Guide. Refer to the discussion on the PL/I Optimizing Compiler.

MNSORT - MUSIC Main Program for Sorting

MNSORT is a general purpose sort program which can sort data records into ascending or descending order according to various types of control fields. Input can be from a disk or tape data set, from the Save Library, or from the input stream. Output sorted records can be punched, printed, or written to tape, disk, or the Save Library. Sorting is done by the subroutine DSORT. FORTRAN NAMELIST input allows the user to specify input and output unit numbers, file names, work units, logical record length, sort control fields, and other parameters. Default values are provided for all parameters.

If you enter only the INPUT='filename' parameter, you can sort the contents of a file and cause it to be replaced with the sorted version.

From a workstation, MNSORT is used by typing EXEC MNSORT or the following:

```
/FILE statements as required
/INCLUDE MNSORT
... cards to be sorted...  (if input unit is 5)
```

The user will then be prompted to enter the sort parameters in NAMELIST form, separated by commas (see below for a description of the parameters). When MNSORT is used from batch, usage is as above except that the NAMELIST parameters must be supplied following the /INCLUDE MNSORT statement.

Parameters

INPUT=n or INPUT='filename' or FILE='filename'
: The input data to be sorted, either a unit number (1 to 13) or a file name (in single quotes). Default is INPUT=1. Abbreviation: IN

OUTPUT=n or OUTPUT='filename' or OUTPUT='*'
: Defines the sort output file, either a unit number (1 to 13) or a file name. OUTPUT='*' means use the same unit or file name as specified by the INPUT parameter (i.e. the sorted output replaces the input file). Default if OUTPUT not specified: OUTPUT='*' if INPUT is a file name; otherwise OUTPUT=2. Abbreviation: OUT.

Notes:

1. An output file is automatically replaced if it already exists.
2. If INPUT is a file name and OUTPUT is not specified, the input file is replaced by the sorted data.
3. If a unit number is used for INPUT or OUTPUT, an appropriate /FILE statement should be used if necessary.

RECLEN=n
: The length of the records to be sorted. If INPUT is a file name, default RECLEN is the record length of the file; otherwise the default is RECLEN=80.

WORK1=n
: Unit number for first (or only) sort work file. Default is 3. If WORK1=0, no work file is used.

WORK2=n
: Unit number for second sort work file. If 0, a second work file is not used. Default is WORK2=4.

Note: The program contains /FILE statements for two work files on units 3 and 4 (temporary files). They are adequate for sorting up to about 550K of data (7000 80-byte records). If necessary, you can change the work file sizes by supplying your own /FILE statements for units 3 and 4:

```
/FILE 3 N(&&TEMP) NEW DELETE SPACE(nnnn)
/FILE 4 N(&&TEMP) NEW DELETE SPACE(nnnn)
/INCLUDE MNSORT
```

where *nnnn* is the desired space in K. The default is 550K. When work files are used, there must be 2 of them (not 1); that is, WORK2 must not be 0.

WRKSIZ=n
: Size in bytes of the main storage sort work area to be used. The maximum is 52000, which is also the default.

FORMAT='xx','xx',...

Formats of the sort control fields. Default is FORMAT='CH' (1 field). The last specification is repeated if necessary. See also FLDPOS, FLDLEN, and ORDER parameters below. The possible formats are: CH (character), BI (binary), DA (date), FI (fixed-point), FL (normalized floating-point), ZD (zoned decimal), PD (packed decimal), CI (case ignore).

FLDPOS=n1,n2,...
Starting column numbers of the sort control fields. Default is FLDPOS=1.

FLDLEN=n1,n2,...
Lengths of the sort control fields. Each value must be 1 to 256. Default is RECLEN, or 256 if RECLEN is greater than 256. Refer to the DSORT routine for other restrictions on FLDPOS and FLDLEN.

ORDER='x','x',...
Sort order for the control fields: 'A' for ascending, 'D' for descending. Default is ORDER='A'. The last specification is repeated if necessary.

TITLE='xxxx...'
Optional information to appear in the heading.

PARMS=n If a nonzero unit number is specified, additional parameters will be read from that unit. The default is PARMS=0.

HELP Requests a display of information on how to use the program.

ECHO Requests a display of the parameter values.

Notes:

1. The same sorting restrictions and efficiency considerations apply to MNSORT as apply to DSORT.

2. The file MNSORT is public and not execute-only, so that the user can replace the /FILE statement for unit 3 or supply replacements for routines SRTIN, SRTOUT, or SRTMSG.

Examples:

1. Assume that data set ABCDFIL1 contains 50-byte records to be sorted in ascending order on two control fields. The first field is a 4-byte fixed-point value starting in column 21; the second is a 10-byte character field starting in column 3. The sorted records are to be written to data set ABCDFIL2.

```
*Go
list sample
*In progress
/FILE 1 UDS(ABCDFIL1) VOL(VOLUME) SHR
/FILE 2 UDS(ABCDFIL2) VOL(VOLUME) OLD
/INCLUDE MNSORT
*End
*Go
sample
*In progress
ENTER SORT PARAMETERS OR HELP
?
reclen=50,format='fi','ch',fldpos=21,3,fldlen=4,10

SRT000  BEGIN SORT.  RECORD LENGTH =  50, AREA =  51944
SRT000  RECORD COUNT =   200
SRT000  NORMAL END OF SORT
JOB TIME    3.42 SERVICE UNITS
*End
*Go
```

2. This example sorts the contents of file TT1 and stores the sorted records into file TT2. The previous file TT2 is replaced.

```
mnsort
*In progress

ENTER SORT PARAMETERS OR HELP
?
input='tt1',output='tt2'

SRT000  BEGIN SORT.  RECORD LENGTH =  80, AREA =  52000
SRT000  RECORD COUNT =   125
SRT000  NORMAL END OF SORT
JOB TIME    2.19 SERVICE UNITS

TT2
REPLACED
*End
*Go
```

DSORT Subroutine

This subroutine sorts fixed-length logical records into ascending or descending order based on one or more sort control fields in each record. The caller supplies a main storage area and, optionally, one or two direct access work files to be used if all records cannot be sorted in the main storage area. The input data set to be sorted, and the output sorted data set, may be any MUSIC sequential files, or user-written input/output routines may be supplied.

If appropriate user-written input/output routines are used, the length of each logical record is limited only by the size of the main storage work area. If sufficient direct access work space is provided, there is no limit on the number of records which can be sorted. Records with identical control fields are output in the same order as input.

Restrictions

1. The total length of the sort control fields may not exceed 256 bytes.
2. The maximum number of control fields is 20.
3. Control fields may not overlap, and each must start within the first 4096 bytes of the logical record.
4. Only the following types of control fields are accepted:
    a. Character (EBCDIC collating sequence) (OS Sort/Merge codes CH and BI).
    b. Internal fixed-point (OS Sort/Merge code FI).
    c. Internal normalized floating-point (OS Sort/Merge code FL). Floating-point control fields must be normalized.
    d. Zoned decimal (OS Sort/Merge code ZD). ZD should be used to sort unsigned numeric fields created using FORTRAN I format.
    e. Packed decimal (OS Sort/Merge code PD).
5. Zoned and packed decimal fields must not exceed 32 bytes in length and are not checked for valid sign, digit or zone codes. Note that non-negative integer fields created using FORTRAN I-format may be treated as zoned decimal fields for sorting.

Usage

DSORT may be called in two ways. All variables except CTLTAB and WKAREA are fullword integers.

```
Mode I:
CALL DSORT(RECLEN,ORDER,CTLLEN,CTLDSP,WKAREA,WRKSIZ,
           INU,OUTU,WORKU1,WORKU2,&n)

Mode II:
CALL DSORT(RECLEN,2,CTLTAB,0,WKAREA,WRKSIZ,INU,OUTU,
           WORKU1,WORKU2,&n)
```

Mode I is used when sorting on a single character-type control field. Mode II is used for the general case of one or more control fields of various types.

RECLEN — Length (in bytes) of each logical record.

ORDER — 0 for ascending order, 1 for descending order.

CTLLEN — Length (in bytes) of the control field. This must be 1 to 256.

CTLDSP — Displacement (in bytes) of the control field from the start of the logical record. This must be 0 to 4095. (A control field at the beginning of the record has a displacement of 0).

CTLTAB — Table specifying the sort control fields for Mode II. The format of this table is described below.

WKAREA — Array to be used as a main storage work area. Usually this is specified as a REAL*8 array.

WRKSIZ Length (in bytes) of the work area WKAREA. The minimum length is 1536 if only one sort control field is used, or approximately 1536 + RECLEN if more than one sort control field is used. However, a larger area should be provided if possible.

INU Unit number of sort input data. This argument is not inspected by DSORT. It is simply passed to the SRTIN routine.

OUTU Unit number of sort output data. This argument is not inspected by DSORT. It is simply passed to the SRTOUT routine.

WORKU1 Unit number or ddname of a direct access work file, or 0. If a ddname is used, it must be followed by a blank (if less than 8 characters long) and enclosed in single quotes. For example, 'SRTWRK '. If only one work file is used (WORKU2=0), then it must be a UDS file, not a file.

WORKU2 Unit number or ddname of a second direct access work file, or 0.

&n This argument is optional. It is the FORTRAN statement number to be transferred to if the sort is unsuccessful. DSORT returns with a code of 0 in register 15 if the sort is successful, and a code of 4 otherwise.

Format of Control Field Table

The table CTLTAB specifying the sort control fields for Mode II is an INTEGER*2 array (that is, halfword integers). The first element, CTLTAB(1), is the number (1 to 20) of sort control fields. The remaining elements, in groups of three, describe the sort control fields as follows:

CTLTAB(3i-1) Displacement (in bytes) of the ith control field from the start of the logical record. This must be 0 to 4095. (A control field at the beginning of the record has a displacement of 0.)

CTLTAB(3i) Length (in bytes) of the ith control field. This must be 1 to 256.

CTLTAB(3i+1) The first byte specifies the type of the ith control field:

0=character
1=fixed-point
2=normalized floating-point
3=zoned decimal
4=packed decimal
5=date (7 character form, i.e. 01JAN90)
6=case ignore

The second byte gives the sort order for the ith control field:

0=ascending
1=descending

An example is given below.

Sort I/O Routines

In addition to using the GULP routines for I/O on the disk work files, DSORT calls three subroutines: SRTIN to read an input logical record, SRTOUT to write an output logical record, and SRTMSG to display a

message. DSORT uses the following calling sequences:

```
CALL SRTIN(RECORD,RECLEN,INU,&n)

CALL SRTOUT(RECORD,RECLEN,OUTU)

CALL SRTMSG(LINE)
```

where RECORD is the logical record to be read or written, RECLEN is the record length (as passed to DSORT), INU is the input unit number (as passed to DSORT), OUTU is the output unit number (as passed to DSORT), and LINE is a 121-character line to be printed. SRTIN does an alternate return (RETURN 1) to indicate end of input.

Default versions of SRTIN, SRTOUT, and SRTMSG are available in the subroutine library. SRTIN and SRTOUT can handle any record length. For special applications, any or all of these routines may be replaced by user-written ones. This is done by including the replacement source or object deck as part of the user's program.

You can suppress all the sort messages by defining a dummy version of SRTMSG such as:

```
SUBROUTINE SRTMSG
RETURN
END
```

DSORT uses 18 and 19 as the GULPDF file numbers for work file I/O. Therefore, the user's program should avoid using these numbers.

Efficiency Considerations

As large a main storage area as possible should be provided. A suggested size is 32000 bytes or more.

The number of direct access work files may be 0, 1, or 2. No work file is needed if all records can be sorted in main storage. Otherwise, the total work space provided must be at least twice the size of the input data set to be sorted. That is, a single work file twice the input size is needed, or two work files each at least the input size. To avoid excessive disk arm movement, two UDS work files should be used only if they are located on different volumes.

For maximum efficiency in sorting a large file, use two UDS work files on different volumes.

If a file is used as a work file, the number of work files must be two. In other words, if a single work file is used, it must be a UDS file, not a file.

Example 1

The following program sorts the card images in data set ABCD$XYZ in ascending order according to the characters in columns 21 to 30 of each card. The resulting cards are punched (unit 7). Mode I calling sequence is used, with two sort work files on different volumes.

```
/FILE 1 UDS(ABCD$XYZ) VOL(MUSIC2) SHR
/FILE 2 UDS(ABCDWRK1) VOL(MUSIC2) OLD
/FILE 3 UDS(ABCDWRK2) VOL(MUSIC3) OLD
/LOAD VSFORT
      REAL*8 WORK(7000)
      CALL DSORT(80,0,10,20,WORK,56000,1,7,2,3,*9)
```

```
      STOP
9     STOP 99
      END
```

Example 2

This example reads cards on unit 5, sorts them according to three control fields, and writes the result on unit 10. Mode II calling sequence is used, with one sort work file. A dummy SRTMSG routine is used to suppress messages. The control fields are:

1. Fixed-point in columns 5 to 8 (ascending order).

2. Floating-point (double precision) in columns 41 to 48 (descending order).

3. Characters in columns 21 to 30 (ascending order).

```
/FILE 1 UDS(&&TEMP) NREC(2000)
/LOAD VSFORT
      REAL*8 WORK(4000)
      INTEGER*2 CTLTAB(10)/3,4,4,Z0100,40,8,Z0201,
     *  20,10,0/
      CALL DSORT(80,2,CTLTAB,0,WORK,32000,5,10,1,0,*9)
      STOP
9     STOP 99
      END
      SUBROUTINE SRTMSG
      RETURN
      END
/DATA

      cards to be sorted
```

SRTMUS Subroutine

The MUSIC subroutine SRTMUS may be used from a FORTRAN program to sort fixed-length records into ascending or descending order according to one or more control fields in each record. SRTMUS performs the sort operation by calling DSORT, the MUSIC generalized sort routine. Refer to the description of DSORT for detailed information about sort work files, main storage areas, and input/output routines.

Syntax

```
CALL SRTMUS(SRTCMD,RECCMD,RETCOD,WRKSIZ,WKAREA,INU,OUTU,WORKU1,WORKU2)
```

Parameters

SRTCMD Image of SORT statement (as for OS Sort/Merge), terminated by a dollar sign character ($).

RECCMD Image of RECORD statement (as for OS Sort/Merge), terminated by $.

RETCOD Integer return code, set to 0 if sort is successful, to 16 otherwise.

WRKSIZ Integer length (in bytes) of the array WKAREA.

WKAREA Main storage work area to be used by DSORT. A recommended size is 32000 bytes or more.

INU Integer unit number of sort input data. If this argument is omitted, unit 1 is assumed.

OUTU Integer unit number of sort output data. If this argument is omitted, the output unit is assumed to be the same as the input unit, and the original data set is replaced by the sorted one.

WORKU1 Integer unit number of a temporary data set to be used as a sort work file. If this argument is omitted, unit 3 is assumed. If 0 is specified, a work file is not used.

WORKU2 Integer unit number of a second sort work file. If this argument is omitted or 0, only one work file is used (WORKU1). Two sort work files should be used only if they are on different volumes.

Restrictions

1. Variable-length records (TYPE=V) may not be used.
2. Sort control formats CH (character), BI (binary), FI (fixed-point), FL (normalized floating-point), ZD (zoned decimal), and PD (packed decimal) are supported.
3. Sort control fields must begin on a byte boundary and be a whole number of bytes in length.
4. The maximum number of control fields is 20.
5. Control fields may not overlap.

Example of SRTMUS

This example generates 20-byte records on unit 1 containing floating-point numbers, then sorts the records into ascending order according to the floating-point numbers and prints the lowest and highest values. File 3 is a sort work file.

```
/FILE 1 UDS(&&TEMP) NREC(2000) LRECL(20)
/FILE 3 UDS(&&TEMP) NREC(1000)
/LOAD VSFORT
      REAL*8 WORK(4000)
      CALL RSTART(123,4567)
      DO 1 M=1,2000
      X=UNI(0)
1     WRITE(1,71) M,X
71    FORMAT(I10,A4)
      ENDFILE 1
      REWIND 1
      CALL SRTMUS(' SORT FIELDS=(11,4,FL,A)$',
     * ' RECORD TYPE=F,LENGTH=20$',K,32000,WORK)
      IF(K.NE.0) STOP 99
      REWIND 1
      DO 2 N=1,2000
      READ(1,71) M,X
      IF(N.EQ.1) WRITE(6,61) X,M
2     IF(N.EQ.2000) WRITE(6,62) X,M
```

```
61    FORMAT('0MINIMUM =',F12.6,' AT M =',I7)
62    FORMAT('0MAXIMUM =',F12.6,' AT M =',I7/
      STOP
      END
```

Using COBOL's SORT Feature

The Sort Feature of ANS COBOL is supported by MUSIC. Using the SORT Statement, the COBOL programmer can sort data records according to specified fields. Refer to the ANS COBOL Language Manual (GC28-6396) for a complete description of the Sort Feature, and a sample program illustrating its use. The following additional remarks apply to COBOL sorts under MUSIC:

1. The sort is performed by DSORT, the MUSIC generalized sort routine described above.

2. A work file must be defined with a ddname of SORTWK01 by using a /FILE statement. This is used as a sort work file by DSORT. If a second sort work file is to be used, define it with ddname SORTWK02. The work files can be UDS or normal MUSIC files, except that a UDS must be used if there is only one sort work file.

   Example using one work file:

   ```
   /FILE SORTWK01 UDS(&&TEMP) NEW DELETE NREC(8000)
   ```

   Example using two work files:

   ```
   /FILE SORTWK01 NAME(&&TEMP) SPACE(500)
   /FILE SORTWK02 NAME(&&TEMP) SPACE(500)
   ```

3. The size of the main storage work area can be specified within the COBOL source program by using the SORT-CORE-SIZE special register. For example, to request an area of 50000 bytes, use "MOVE 50000 TO SORT-CORE-SIZE". Negative numbers and the special value +999999 are not supported.

4. Variable-length records may not be sorted.

TAPUTIL

TAPUTIL is a tape utility program which can be used to dump or summarize selected files from tape, and copy files from one tape to another. All record formats and block sizes can be processed, and blocks may be of different lengths. Blocks can be dumped in both character and hexadecimal form.

Usage

TAPUTIL is run as a MUSIC batch job, with the tape or tapes specified on /FILE statements. The input tape (the tape to be read) is normally defined as unit 1. The output tape, if required, is normally defined as unit 2. An output tape is used only if the COPY parameter is specified. The control statement setup is as follows:

```
/FILE 1 TAPE VOL(volume1) RECFM(U)
/FILE 2 TAPE VOL(volume2) RECFM(U)  (if a second tape is used)
/INCLUDE TAPUTIL
parameter statement (see below)
```

The /FILE statements for the tape(s) must specify RECFM(U), since this program processes raw blocks of data of any length. The program is controlled by various keyword parameters (in Fortran NAMELIST form) on the parameter statement. These parameters, described below, are separated from each other by commas.

Parameters

The following may be specified on the parameter statement, separated from each other by commas. The default column lists the parameter values which are automatically assumed if the parameter is not specified.

STATS — Print statistics about each file (number of blocks, lengths of blocks, etc.) The default is no STATS.

SL — Print any standard labels found. The default is no SL.

COPY — Copy files from the input tape to the output tape. The default is no COPY.

IN=n — Unit number of input tape. The default is 1

OUT=n,f — Unit number (n) of output tape and starting file number (f) in the output tape. The defaults are 2,1.

PRINT=n — Print the first *n* blocks from each file, in character form. PRINT='ALL' may be specified. The default is 0.

DUMP=n — Dump the first *n* blocks from each file, in character and hexadecimal. DUMP='ALL' may be specified. The default is 0.

FILES=n — Process the first *n* files from the input tape. If not specified, two tape marks will be regarded as the end.

SELECT=n,m,... — Process only files *n, m,* ... from the input tape. Up to 100 file numbers may be specified.

They must be in ascending order.

Notes

1. Record format U must be specified on the tape /FILE statements.

2. A *file* of a tape is considered to be the data between two consecutive tape marks (or between the beginning of the tape and the first tape mark). The files of a tape are numbered 1, 2, 3, etc. MUSIC has no *skip to file* support, so files are skipped by reading through them.

3. The program assumes that two tape marks (with no data blocks between them) indicate the end of the input tape. If the tape does not end with two tape marks, the program will continue reading to the end of the reel. This is not very popular with operations. It is recommended that the FILES=n parameter be used to limit the number of files processed. This parameter can also be used to skip over double tape marks in the middle of a volume. However, the FILES and SELECT parameters must not be used together.

4. Users are reminded that MUSIC tape jobs must be submitted by using the SUBMIT command described earlier in this guide in *Chapter 3. Using Batch*.

5. The STATS, SL, and copy parameters must not immediately follow the OUT or SELECT parameters.

Examples

1. Dump the first 5 blocks from every file of a tape.

```
/FILE 1 TAPE VOL(MYTAPE) RECFM(U)
/INC TAPUTIL
DUMP=5
```

2. Print the first 10 blocks of each of files 1, 5, and 7.

```
/FILE 1 TAPE VOL(MYTAPE) RECFM(U)
/INC TAPUTIL
PRINT=10,SELECT=1,5,7
```

3. Copy TAPEX to TAPEY.

```
/FILE 1 TAPE VOL(TAPEX) RECFM(U)
/FILE 2 TAPE VOL(TAPEY) RECFM(U)
/INC TAPUTIL
COPY
```

4. Dump the entire contents of a tape, print statistics, and any standard labels.

```
/FILE 1 TAPE VOL(XXXXXX) RECFM(U)
/INC TAPUTIL
STATS,SL,DUMP='ALL'
```

5. Copy files 3 and 4 from TAPEIN to file 6 and 7 on TAPEOT.

```
/FILE 1 TAPE VOL(TAPEIN) RECFM(U)
/FILE 2 TAPE VOL(TAPEOT) RECFM(U)
/INC TAPUTIL
COPY,OUT=2,6,SELECT=3,4
```

Archiving and Retrieving User Data Set (UDS) Files

The user can cause a set of User Data Set (UDS) files owned by the user to be copied to a magnetic tape by using the UDSARC program. The referenced data sets on disk are not altered by this copy operation. Options are available to archive specific data sets, a group, or all of them to tape. (To archive files, refer to the writeup on the ARCHIV utility.)

The UDSRST program can be used to retrieve data sets from tapes created by UDSARC.

The DSCHK utility is available to list the names of the files contained on a tape created by the UDSARC program and at the same time, double-check the validity of the information stored on the tape.

The user should note that these three utilities must be submitted to batch since they use magnetic tape.

UDSARC Program

The following illustrates the control statement set up for the UDSARC program:

```
/FILE 1 TAPE BLK(nnnn) RSIZ(80) VOL(vvvvvv)
/INCLUDE UDSARC
UNITS=1
... dump specification statements (see below)
```

Dump Specification Statement:

```
VOL='vol1'[,'vol2'][,'vol3'][,DSN='data set name']
                           [,CODES='userid'     ]
```

```
Examples:     VOL='MUSIC1',DSN='ABCD1234'
              VOL='MUSIC1','MUSIC2',CODES='ABCD'
              VOL='MUSIC2',DSN='ABCDX..'
```

One or more dump specification statements may be used. Each statement must contain a VOL parameter and one of either a DSN or CODES parameter. Abbreviations V, D and C may be used. Commas are used to separate the parameters on each dump specification statement.

VOL specifies a list of volume names to be searched for the data sets. One or more volume names may be given.

DSN specifies the name of the data set to be archived. The form DSN='ABCDX..' may be used to indicate that all data sets on the named volumes starting with the letters ABCDX are to be archived.

CODES parameter causes all the user's data sets on the given volumes to be archived. Specify the ownership id (userid without subcode).

Retrieving UDS Files: UDSRST

UDS files can be retrieved from the tape prepared by the UDSARC program. Each data set is retrieved from tape by running a separate batch job. If the receiving data set does not exist, it must be allocated using the original logical record size (RSIZ) and number of records (NREC). The parameters VOL and DSN give the volume and data set name at the time the data set was archived. These parameters are used to locate the specific copy on tape.

The following illustrates the control statement set up:

```
/FILE 1 TAPE BLK(nnnn) RSIZ(80) VOL(vvvvvv) SHR
/FILE 2 UDS(receiving data set name) VOL(volume) OLD
/INCLUDE UDSRST
IN=1,OUT=2,VOL='original volume',DSN='original data set name'
```

DSCHK Program

This program checks the contents of a tape created by UDSARC and can list the data set names stored on it if the INFO=TRUE parameter is used. The following illustrates the control statement set up for running the DSCHK program:

```
/FILE 1 TAPE BLK(nnnn) RSIZ(80) VOL(vvvvvv) SHR
/INCLUDE DSCHK
UNITS=1,INFO=TRUE
```

Contents of Information Records

In a dump produced by UDSARC, each data set is preceded by a *DS1 record and a *DS2 record. The *DS1 record has the volume name, the data set name, and the date and time of the dump. The *DS2 record contains the following information:

Column 8	B for backup, N for no backup.
9-12	Device type.
13-15	Data set organization.
16-19	Record format.
20-25	Block size.
26-31	Logical record length.
32-37	No. of tracks (no. of physical blocks if FBA device).
38-40	No. of extents.
41-44	No. of blocks per track (32 if FBA device).
45-48	Key length.
49-53	FBA physical block length (0 if not FBA device).

UTIL

UTIL provides facilities for listing, punching, copying, and merging data on cards, card images, tape or disk. It can insert sequence numbers, translate from BCD to EBCDIC and vice versa, and perform other editing functions. Multiple copies can be punched or printed. The program can read and write logical records up to 133 bytes in length, and can supply information on the size and characteristics of MUSIC disk data sets.

Usage

Typically UTIL is run using a control statement set up similar to that shown below:

```
/FILE statements   (if required)
/INCLUDE UTIL
...control statements and data...
```

If more than one copy of punched or printed output is requested (via the control statements $PUNCH=n, $LIST=n or $COPIES=n described below), then unit 4 is reserved as a work file and thus the user cannot give a /FILE statement for this unit number. If copies involve more than about 4000 records each, then the user should provide a temporary file on unit 4 that will be large enough to hold one copy. If multiple copies are not requested, then the user may supply as many as 4 /FILE statements referring to UDS files.

For data records in the input stream (unit 5) or accessed by /INCLUDE, the record length is 80. For data records accessed by $INPUT=m, the record length is that of the file specified on the /FILE statement for unit *m*, to a maximum of 133.

Control Statements

Control statements normally have $ in column 1 and may not contain embedded blanks. In the following description, *n* is an unsigned decimal number, 1 to 8 digits in length. A quote within a data string must be represented by two single quotes. With no control statements, the input deck is simply listed.

A user data record may have the control character $ in column 1 provided a UTIL control statement name does not occur starting in column 2. (See also $NOCTL, $INPUT and $CHAR below.)

$BACKSPACE=n
: Performs a backspace operation on MUSIC I/O unit number *n.*

$BATCH
: Causes line numbers, indentation, and page headings to be produced by default. The page heading includes the date and page number, unless suppressed by $NOPAGE.

$BCD=n
: Convert from EBCDIC to BCD and punch *n* copies. Default is n=1. (Number of copies is ignored if already specified.) If both conversions are requested, BCD to EBCDIC is done first.

$CHAR=x
: Use *x* (may be any character) as the control character instead of $.

$CONV
: Convert from BCD to EBCDIC as cards are read.

$DOUBLE
: Double space listing.

$EBCDIC=n Equivalent to $CONV followed by $PUNCH=n.

$END Treated as end-of-file.

$ENDFILE=n Write an end-of-file mark on MUSIC input/output unit number *n.*

$GANG
n
card The above sequence punches *n* copies of *card.*

$INDENT=n In the listing, indent *n* spaces before the record. *n* must be 0 to 40. Default is n=0 (n=20 if $BATCH is used).

$INFO Causes a description of UDS files to be printed at the end of the program. Temporary (that is, unnamed) disk data sets are not included.

$INFO=ALL Same as $INFO, except that temporary UDS files are included.

$INPUT=m Causes reading to immediately transfer to unit *m.* At end-of-file on this unit, reading switches back to unit 5. Input logical records shorter than 133 bytes are filled out with blanks on the right. *m* may be 1, 2, 3, 4, 5, 9, 11, 12, 13, 14, or 15. More than one $INPUT card may be used, and the same unit number may be repeated; thus, two or more data sets may be merged. For units 5 and 9, control statements are accepted (e.g. $INPUT=5 may be used to switch from 9 to 5, or $END may be used to stop the program from unit 9), but for other units any control statements are treated as data.

Input is from unit 5 (the input stream) until the first $INPUT=m statement is encountered. The record length for the input stream, or for any /INCLUDE'd file, is 80.

Note: If a $OUTPUT or other control statement is to apply to the data read by a $INPUT statement, it must **precede** the $INPUT statement.

$LIST=n
$COPIES=n Either of these control statements is used to resume or begin listing, and produce *n* copies of printed output. *n* (and the equal sign) may be omitted, in which case n=1 is assumed.

$LIT=('string1',starting col,'string2',starting col)
Specifies up to 2 character strings to be inserted in each card. For example, $LIT=('ABCD',73) puts ABCD in columns 73-76 of each card.

$NAME=a...an...n
Specifies identification *a...a* and sequence numbers (initial value and increment both *n...n)* for columns 73-80. *a...a* and *n...n* must together have a length of 8, and the length of *n...n* must not be zero. *a...a* may not contain blanks or end with a digit.

$NOBCD Suppress conversion from EBCDIC to BCD. (Punching continues unless suppressed by $NOPUNCH -- the same applies to $NOEBCDIC.)

$NOCTL Causes any subsequent control statements to be treated as data.

$NOCONV Either control statement suppresses conversion from BCD to EBCDIC. (This is the default.)
$NOEBCDIC
$NOLIST Suppress listing.

$NOLIT Suppress insertion of character strings.

$NOPAGE Suppress date and page numbers from the page heading line. (Meaningful only if $BATCH is used.)

$NOPUNCH Suppress punching. (This is the default.)

$NOSEQ Suppress sequence numbers.

$NUMBER Either of these control statements is used to cause a skip to a new page and number the
$NUM following printed lines, starting at 1.

$OUTPUT=m Specifies that output is to be written on unit *m*, and implies 1 copy. *m* may be 1, 2, 3, 4, 6, 7, 10, 11, 12, 13, 14, or 15. Output logical records are truncated or blank-padded on the right to conform to the receiving data set.

The combination of $NOLIST and $OUTPUT=6 can be used to print a file which has carriage control information as the first character of each record.

Use the $PUNCH=n option to get multiple copies of the output. If used this way, the $PUNCH card must precede the $OUTPUT card.

Note: If a $OUTPUT card is to apply to one or more $INPUT cards, it must precede the $INPUT cards.

$PUNCH=n Write the output on unit 7, producing *n* copies (for example, punch *n* copies of an input deck). *n* may be omitted, in which case n=1 is assumed. Only the first $PUNCH card is used to set the number of copies.

$REWIND=n Issues a rewind for MUSIC I/O unit number *n.*

$SEQ=(starting-col,length,initial-value,increment)
Specifies sequence numbers to be inserted. Length must be 1 to 8. $SEQ is equivalent to $SEQ=(77,4,1,1).

$SEP=n Punch *n* separator cards between each copy (n may be 0). Separator cards have 7-8-9 punches in columns 1-80. Default assumption is 5 separator cards. Separator cards are suppressed if the output unit is not 7.

$SINGLE Single space listing. (Default is $SINGLE.)

$SKIP
$EJECT Either of these control statements will cause a skip to a new page.

$SPACE=n Insert *n* blank lines in the listing. Default is n=1. If fewer than *n* lines remain on the page, effect is same as $EJECT.

$STATS Prints input and output record counts. The input count does not include control cards and the output count does not include extra copies of punched output or separator cards.

$SUPPRESS
$NONUM Either control statement suppresses line numbers.

$TERM Reverts to standard defaults, i.e. no line numbers, no indentation, and no page headings. $TERM is assumed at the start of the program.

$TITLE='string'
Skip to a new page and print specified title at top of each page. (Meaningful only if

$BATCH is used.)

$* Comment (ignored).

Examples

Copy a sequential file from one UDS to another (unit 1 to unit 2). The logical record length is assumed to be 133 or less.

```
/FILE 1 UDS(ABCDXXX1) VOL(VVVVVV) SHR
/FILE 2 UDS(ABCDXXX2) VOL(VVVVVV) OLD
/INCLUDE UTIL
$NOLIST
$STATS
$OUTPUT=2
$INPUT=1
```

Print a file called ABC with line and page numbers

```
/INCLUDE UTIL
$BATCH
/INCLUDE ABC
```

Card to tape with sequence numbered listing

```
/FILE 1 TAPE BLK(800) RSIZ(80) VOL(MYTAPE) OLD
/INCLUDE UTIL
$OUTPUT=1
$NUMBER
$NOCTL
(DATA CARDS)
```

Card to disk data set, suppressing listing

```
/FILE 1 UDS(UUUUSIII) VOL(MUSIC2) OLD
/INCLUDE UTIL
$OUTPUT=1
$NOLIST
$NOCTL
(DATA CARDS)
```

Print and punch from magnetic tape

```
/FILE 1 TAPE BLK(800) RSIZ(80) VOL(MYTAPE) SHR
/INCLUDE UTIL
$OUTPUT=7
$INPUT=1
```

Listing a disk UDS file with line and page numbers

```
/FILE 1 UDS(UUUUSIII) VOL(MUSIC2) SHR
/INCLUDE UTIL
$BATCH
$INPUT=1
```

Print 10 copies of a file whose records are 100 bytes long

```
/FILE 1 NAME(FILENAME)
/INCLUDE UTIL
$COPIES=10
$INPUT=1
```

Reproduce a card deck:

Add identification in columns 73-76 and sequence numbers in columns 77-80. No listing to be produced.

```
/INCLUDE UTIL
$NOLIST
$NAME=CARD0001
$OUTPUT=7
(DATA CARDS)
```

Punch 500 copies of a card

```
/INCLUDE UTIL
$GANG
500
(CARD)
```

Obtain information about a disk data set:

```
/FILE 1 UDS(UUUUSIII) VOL(MUSIC2)
/INCLUDE UTIL
$INFO
```

VIEW

VIEW allows the viewing of files of any record length or type and any size on a full-screen workstation. Files can be displayed in hexadecimal. ASCII mode is available to view ASCII files on a 3270-type workstation. In this mode printable ASCII characters are translated to their EBCDIC equivalents. Locate and Find functions are supported.

Summary of Commands

AScii [on|off|flip]	HUNT string	Search string
Bottom	Locate string	SHow [on|off]
CANcel	LEft n	STatus
CAse [Ignore|Respect]	LIne n	STORE fn [Append|Replace]
CEnter n	MArk [m n|?]	TAG [on|off|flip]
COlumns [on|off|flip]	MARKER x	TEXTLc
DEFine PFn string	Next n	TEXTUc
Down n	LLine n	TIme
ECHO [NAME|WORD|text]	NUMber [on|off|flip]	Top
ECHOX [NAME|WORD|text]	QQuit	UIO [on|off|flip]
END	RAnge m n	UNMark
Find string	RETrieve	Up
HELP	RIght n	Users
HEX [on|off|flip]	RUler [on|off|flip]	VHEX [on|off|flip]
HHEX [on|off|flip]	SCROll n	Zone m n

Function Keys

F1: Help	Get help on how to use VIEW.
F3: End	Terminate the current VIEW.
F5: CEnter	Place the data line pointed to by the cursor at the center of the data display area.
F7: Up	Move by the scroll value toward the top of the file.
F8: Down	Move by the scroll value toward the bottom of the file.
F9: Locate	Repeat the previous locate command.
F10: LEft	Move by the scroll value toward the first column of the file.
F11: RIght	Move by the scroll value toward the last column of the file.
F12: RETrieve	Retrieve the previous command entered.

Commands

AScii [on|off|flip]	Turns on or off ASCII translation. If the file being viewed is an ASCII file, "ascii on" will allow the display of the standard printable characters.

Bottom — Displays the last full page (screen) of the file.

CANcel — Terminates the current view.

CAse [Ignore|Respect]
Defines the case for searching strings. When RESPECT, the result of the search depends upon the setting of TEXTLC or TEXTUC. If TEXTUC is in effect, then your input text is translated to upper case prior to the search. If TEXTLC is in effect, then your input string is not translated to upper case. In both circumstances, the text in the file is compared "as-is". When IGNORE, the searching ignores case, both in your input string and in the file. Any combination of upper and lower case matches the searched text.

CEnter n — Places the line defined by *n* at the center of the display area. If *n* is not specified, the line pointed to by the cursor is centered.

COlumns [on|off|flip] — This command places a ruled line in the message area that corresponds to the columns being displayed.

DEFine PFn string — Defines the function key specified by *n* to be *string.* If no string is specified, the current definition is place in the command area, ready for modification. If the string specified is an asterisk (*), the key reverts to the default definition.

Down n — This command moves down the viewing window by *n* lines, or when *n* is not specified, it moves down the viewing window by the scroll value.

ECHO [NAme|WORD|text]
Echoes to the command area the text pointed to by the cursor. When the key word "NAME" is used it echoes the viewed file name. When the keyword "WORD" is used the text being echoed is truncated after the first word of the text. When text follows the echo command that text is echoed to the command area.

ECHOX [NAme|WORD|text]
Identical to ECHO except that this command places the hexadecimal version of the text in the command area

END — Terminates the current view.

Find string — Finds the next occurrence of the specified *string.* The search proceeds from the line pointed to by the cursor plus 1. The string to be found must be found at the start of the defined locate zone. See Locate, Zone and RAnge.

HELP — Provides help on how to use the view facility.

HEX [on|off|flip] — Turns on vertical hex display of the data on the screen.

HHEX [on|off|flip] — Turns on horizontal hex display of the data on the screen.

Hunt string — Same as FIND except that the FIND begins at the top of the file.

Locate string
'string'
X'string' — Locates the next occurrence of the specified *string.* The search goes from right to left, and downward, from the line and record column pointed to by the cursor. The cursor is placed at the found text. The view of the file on the screen changes in accordance with the result of the search. If the found text is not on the currently displayed lines and columns, the viewing window adjusts as required. If the search fails, a message is posted informing you that the search has gone to end-of-file,

and the current line and cursor position remains unchanged. The search can be limited horizontally (column-wise) by setting the zone. Setting the range limits searches within the lines defined by the range.

You can specify strings to be located by delimiting them with single (') or double (") quotes. Though quotes are normally not required, they are necessary when leading or trailing blanks are part of a string or when the string to be located is expressed as a hexadecimal string (LOCATE X'string').

Examples:

`l abc`	locate the string -- abc
`l ' abc '`	locate the string -- abc preceded and followed by a blank.
`l "don't"`	locate the string -- don't
`l x"c1c2c3"`	locate the hexstring -- c1c2c3 (c'abc')

Illegal strings

`don't`	strings with imbedded quotes should be quoted by the alternate quote. In this case: "don't".
`'abc`	missing ending quote.
`'abc"`	the first quote must be matched by the same same type quote at the end of the string.
`x'f1fm'`	the string type (x) requires that all hexadecimal digits be in the range a-f and 0-9.
`x'c1c'`	the string type (x) requires that all hexadecimal digits be paired. In this example the string has a trailing "c" which is not paired.

LEft n — This command moves the viewing window left by *n* lines, or when *n* is not specified, it moves the viewing window left by the scroll value.

LIne n — Displays the page (screen) with *n* as the first line. "n" can be a "." (period) indicating the first line of a marked group.

LLine n — Defines the row of the displayed data area to place the "located text" after a locate, where the "locate text" was not on the currently displayed screen. By default such searches will place the located text at the center of the screen. *n* can be any positive integer or any of the following keywords PAGE, HALF, CENTER, CURSOR or MAX.

MArk [m n|?] — Marks the line pointed to by the cursor. A group of lines can be marked. When integer values are used as parameters those lines are marked. "?" used as a parameter, displays in the message area what lines are marked and how many.

MARKER x — used to define the mark character to be used to flag marked lines.

Next n — This command moves the viewing window down by *n* lines, or when *n* is not specified, it moves down by the scroll value.

NUMber [on|off|flip]
: Turns line numbering on or off.

QQuit
: Terminates the current view.

RAnge m n
: Sets the range, start and end lines, for find, locate, top, bottom, up and down commands. *m* is the starting line and *n* is the ending line. If only one number is specified it is taken as *n* and *m* is defaulted to 1. In other words, when only 1 number is specified, the range is taken to be the first "number" lines in the file. RA MAX is used to reset it to the entire file length (all lines). Top of file and bottom of file messages are changed to top of range and bottom of range, when the range is a subset of the file size.

  Examples:

  `RA 10 30` only lines 10 through 30 can be displayed and searched through.
  `RA max` sets the range to the file size.

RETrieve
: Echoes to the command buffer the previous command entered.

RIght
: This command moves the viewing window right by *n* lines, or when *n* is not specified, it moves right by the scroll value.

RUler [on|off|flip]
: This command places a ruled line in the message area which corresponds to the columns being displayed.

SCROll n
: This is an alternate method of setting the scroll value. Normally the scroll value is set by typing in the scroll area directly. Valid values for scroll is any valid positive integer, and the key words *max*, *page*, *half*, and *cursor*.

Search string
: Same as LOCATE except that the search begins at the top of the file.

SHow [filename]
: Display the first two lines of *filename* at the bottom of the screen. This is usually used to display function key definitions. "SHOW OFF" will display the default function key definitions.

STatus
: Displays the current time, date, service units used, number of users current signed on, and the session id number.

STOre fn [Append|Replace]
: Stores the marked lines in the specified file. If append is specified, the marked lines will be appended to the file. If replace is specified the file will be purged and a new file with the marked lines will be created. If neither option is specified, replace is assumed. In this case you will be prompted if the file already exists.

TAG
: Displays the tag text of the viewed file.

TEXTLc
: When specified, the command area is not translated to uppercase. Therefore searching is performed on an upper and lower case basis.

TEXTUc
: When specified, the command area is translated to uppercase. Therefore searching is performed on uppercase basis only.

TIme
: Same as STATUS.

Top	Displays the first full page (screen) of the file.
UIO [on|off|flip]	Turns unformatted I/O on or off. Unformatted I/O refers to reading in the data from the file as 512 byte chunks. These chunks are exact representations of the file as it is recorded on disk. Some files may have been created or written using unformatted I/O. This is detected by VIEW and UIO is automatically turned on. UIO off is invalid for such files since they can not be read sequentially.
UNMark	Unmarks the marked lines.
Up	This command moves the viewing window up by *n* lines, or when *n* is not specified, it moves up by the scroll value.
USers	Same as STATUS.
VHEX [on|off|flip]	Turns on vertical hex display of the data on the screen.
Zone m n	Sets the zone, start and end columns, for find and locate commands. *m* is the starting column and *n* is the ending column. If only one number is specified it is taken as *n* and *m* is defaulted to 1. Z MAX is used to reset it to the entire record length (all columns). **Examples:** `z 10 30` the located string must occur within columns 10-30. `z 50` the located string must occur within columns 1-50.

The Scroll Area

This field on the extreme right of the command area, is used to define the scroll value for right/left and up/down movement. Valid entries in this field are any positive integer or any of the following keywords: PAGE, HALF, CENTER, CURSOR, and MAX. You can change the scroll value by overtyping a new value in the field or you can use the "SCROLL n" command. Commands that use the scroll value are:

UP (F7)	scroll towards the top of the file
DOWN (F8)	scroll towards the bottom of the file
LEFT (F10)	scroll towards the first column of the file
RIGHT (F11)	scroll towards the last column of the file

The use of MAX in the scroll field scrolls to the top, bottom, left margin, or right margin, depending on which of the function keys above you press next. The previous scroll value reappears and takes effect after the interaction, that is MAX is always temporary.

The use of CURSOR scrolls to the data line pointed to by the cursor or if the cursor is off the data area by a PAGE value. For example, if the cursor is pointing to line 50 of a file, the DOWN key places line 50 at the top of the data area on the screen.

Return Codes

0	normal termination.
-1	not a 3270 workstation.

<-1 not enough work area available, absolute of the return integer is the required area in bytes.

1-100 file error occurred.

>100 FSIO error occurred.

VMPRINT

The VMPRINT program is used to print the contents of MUSIC files on a batch printer without the necessity of submitting a batch job. The data writes to a VM virtual printer for subsequent printing by VM. The program has the ability to direct files to remote printers or to other VM systems via RSCS (the Remote Spooling Communication Subsystem of VM).

Usage

```
VMPRINT  file1[,file2,...filen][,CLASS='x'][,TO='yyy'][,CC='zzz']
```

where:

file1 — is the name of the MUSIC file to be printed. One or more files can be specified per command.

CLASS='x' — specifies the VM output class for this printer file (1 character long). The default is class A.

TO='yyy' — specifies the RSCS node ID name, referred to as the destination (up to 8 characters). This is the tag information, which can be the remote printer assigned name or the name of another VM system. When TO= is present, the print file is spooled to RSCS; when TO= is absent, the print file is spooled to SYSTEM (the default).

CC='zzz' — *zzz* can be either YES or NO indicating whether the file to be printed contains control characters in the first column which will control the printing of the file. Acceptable carriage control characters are blank, 0, +, -, and 1. If CC= is not present, then the following defaults are assumed:

- files with a fixed or fixed compressed record format, whose record lengths are not 121 and not greater than 132, do not contain carriage control characters.
- all other files do contain carriage control characters.

The operands are entered on the same line as the command and can be separated by any number of blanks or commands. Any options specified must be enclosed in single quotes.

Examples:

```
VMPRINT MYFILE
VMPRINT FILE1, FILE2, CLASS='A', CC='YES'
```

Program to Write Zeroes to a File

The ZERO.FILE utility program provides an efficient way of writing binary zeroes to every block of a file or user data set. Usage is as follows:

```
/FILE 1 UDS(dsname) VOL(volume) OLD
/INCLUDE ZERO.FILE
START=n,END=m                          <--- optional

or

/FILE 1 NAME(filename) OLD
/INCLUDE ZERO.FILE
START=n,END=m                          <--- optional
```

In the case of a file, the currently allocated space is set to zero. Each 512-byte block is zeroed, therefore this program should not be used for a file which has a record format FC, V, or VC and is later to be read sequentially (error 46, "file cannot be read sequentially" would result).

Appendixes

Appendix A. IIPS/IIAS

Interactive Instructional Systems

The MUSIC Interactive Instructional Presentation and Authoring System (IIPS/IIAS) is an implementation of the IBM Interactive Instructional Presentation System (IIPS) (Program Product 5668-012) and Interactive Instructional Authoring System (IIAS) (Program Product 5668-011). It provides a real-time training and instructional capability for course creation and presentation. The IIPS and IIAS combine the best features of IBM's previous training and instructional programs: the Interactive Instructional System (IIS) and Coursewriter III.

Facilities provided by the IIPS and IIAS include:

1. Administrator commands
2. Report programs
3. Course creation
4. Course execution

Administrator commands provide a means to monitor and control administrative, maintenance, and operational functions of the instructional system. Information relating to the execution and maintenance of course material can be recorded.

A number of programs are provided to process these recordings, producing a variety of statistical summaries.

Instructional materials (courses) prepared by the author are entered into the system from a workstation. The courses may be written in the Coursewriter Language or using the IIAS Course Structuring facility. Once the courses have entered, they are available to the students registered to them.

Course execution is carried out in a conversational manner between a student at a workstation and the instructional system control program. The instructional system presents material to the student, evaluates responses, and, based upon the responses, determines the student's path through the material created by the author.

Appendix B. LEARN Program

If your installation has installed the MUSIC Interactive Instructional Presentation System (IIPS), several computer-assisted instruction (CAI) courses are available to familiarize the user with the basic concepts of the MUSIC system for TTY-type terminals. The names of the courses are:

1. MUSIC (Introduction to MUSIC)
2. UPDATE (Introduction to MUSIC Line Editor)
3. EDITOR (Introduction to MUSIC Context Editor)
4. SCRIPT (Introduction to MUSIC/SCRIPT)

Course Sign On

To sign on to a particular course, enter the word LEARN anytime you are in *Go mode. The system will prompt you with a message

```
ENTER STD#/CNAME OR LIST OR HELP
```

Three responses are valid at this time:

1. Enter "student/cname", where *cname* is one of the course names listed above. For example, enter "student/script" if you want to take the MUSIC/SCRIPT course. (LEARN recognizes *student* as an authorized student number.)

2. Type LIST to get a list of all the MUSIC CAI courses available.

3. Type HELP to get a description of the usage of special keys on your workstation.

Whenever you sign on any one of the courses, the system will create a file called @LEARN for you. This file is used to record information about your progression in the course. The next time when you sign on the same course, it will start from where it left off. Should you want to start at the beginning of the same course the next time when you sign on it, you can purge the file by using the command PURGE @LEARN before you enter LEARN.

If you are taking a course different from the one you took previously, the contents in the file @LEARN will be replaced by the information about your progression in the current course. Therefore, if you sign on the previous course the next time, the course will start at the beginning.

Course Sign Off

Should you want to get out of the course while you are in the middle of it, type SIGN OFF anytime while the system is waiting for your answer to a question. The workstation will then return to *Go mode.

If you have come to the end of the course, the system will issue a message `END OF COURSE PLEASE SIGN OFF.` At this time, enter SIGN OFF to exit from the course and return to *Go mode.

Index

Index

B

C

D

E

F

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Table of Contents

Figures

MUSIC/SP
User's Reference Guide
August 1995

READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that the MUSIC Product Group may use or distribute whatever information you supply in any way it believes appropriate without inccurring any obligation to you.

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, institution, mailing address and date:

_____

_____

_____

_____

What is your occupation?_____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation.

MUSIC/SP User's Reference Guide (August 1995)

Reader's Comment Form

fold and tape please do not staple fold and tape

Place stamp here

MUSIC Product Group
McGill Systems Inc.
550 Sherbrooke St. West
Suite 1650, West Tower
Montreal, Quebec H3A 1B9
CANADA

fold and tape please do not staple fold and tape