# MUSIC/SP

**Version 5**

**Release 1**

# User's Reference Guide

April 1996

**Seventh Edition (April 1996)**

This edition applies to Release 1 of Multi-User System for Interactive Computing / System Product (MUSIC/SP) Version 5, and to all releases of this product until otherwise indicated in new editions or Technical Newsletters. MUSIC/SP Version 5 is published and licensed by McGill Systems Inc.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to: MUSIC Product Group, McGill Systems Inc., 550 Sherbrooke St. West, Suite 1650, Montreal, Quebec, Canada H3A 1B9. Fax: (514) 398-4488.

# Preface

## About this Manual

This manual details the user components of the MUSIC/SP system. No one needs to be familiar with all the information in this publication, as it is intended for advanced users.

First time users of MUSIC/SP should read the publication *MUSIC/SP Guide for New Users*.

Users of Personal Computers should also consult the *MUSIC/SP Personal Computer Workstation User's Guide* for information on dial-in access. Additional information about Personal Computers is found in *Chapter 2. Workstations*.

## What is MUSIC/SP?

MUSIC/SP (Multi-User System for Interactive Computing/System Product) is a multi-user, multi-function, interactive system complete with a collection of application programs, compiler interfaces, and utilities. The collective goal is to provide a high-performance, cost-effective, and manageable computing environment. This environment can include IBM Personal Computers connected to MUSIC/SP running on an IBM mainframe processor.

Running as a single virtual machine, MUSIC/SP can support over a hundred concurrent users performing such diverse activities as problem solving, program development, accessing the Internet, accessing a CWIS, file editing, personal computer support, electronic office functions, and job submission with output retrieval to such systems as MUSIC/SP, DOS/VSE, and CMS batch.

This manual describes the usage of the MUSIC features and facilities available to the general user. Information on workstations, command language, job processing, and processor usage is included.

## Chapter Overview

**Chapter 1. Introduction**

Provides an introduction to the MUSIC/SP system. Key concepts of the command language and file system are given. Also included in this chapter, is information about the menu-driven facilities available with MUSIC/SP.

**Chapter 2. Workstations**

Discusses how to use the various types of workstations that connect to MUSIC. Control keys specific to a particular workstation are described. This chapter includes information on the various ways of connecting IBM Personal Computers to MUSIC.

**Chapter 3. Using Batch**

Discusses how to submit batch jobs to MUSIC batch and other systems.

**Chapter 4. File System and I/O Interface**

Discusses the Save Library and User Data Sets. Tape and disk usage is explained. The unit number convention is described. Usage of VSAM files under MUSIC is explained.

**Chapter 5. MUSIC Commands**

Discusses MUSIC command language. Syntax and examples are given for each command. (The REXX command executor is documented in *Chapter 8. Processors*.)

**Chapter 6. MUSIC Job Control Statements**

Discusses Job Control Statements. Syntax and examples are given for each statement.

**Chapter 7. Using the Editor**

Discusses how to change files with the Editor. Syntax and examples are given for the Editor commands. The Editor can also be used to search files for specific information without modifying them.

**Chapter 8. Processors**

Discusses each compiler and loader available to the MUSIC user. The linkage editor is also explained. Concepts of source, object, and load modules are given. The REXX command executor is documented in this chapter.

**Chapter 9. System Subroutines**

Discusses the many MUSIC system subroutines callable from high-level languages.

**Chapter 10. Utilities**

Discusses the system utilities that can be used to copy, list, backup, and restore a file. PROFILE (the utility to change your password, and so forth) is also described.

**Appendix A. IIAS/IIPS**

Describes the IBM Interactive Instructional Systems IIAS and IIPS.

**Appendix B. LEARN Program**

Describes the online courses available with MUSIC/SP.

**Index**

Useful in locating a specific item or topic. Can be used when tracking down a word or abbreviation that you are not familiar with.

# MUSIC/SP Publications

The following is a list of all the current MUSIC/SP publications. These hardcopy publications can be ordered through the MUSIC Product Group. Online versions (softcopy) of the user publications can be accessed with the MUSIC/SP command called "MAN".

- *MUSIC/SP Administrator's Guide* (April 1996), describes how to install and operate MUSIC/SP.
- *MUSIC/SP Administrator's Reference* (April 1996), describes the internals of MUSIC/SP; utility programs and supervisory commands; gives detailed storage estimates; and documents console messages.
- *MUSIC/SP User's Reference Guide* (April 1996), describes how to use MUSIC/SP; its command language; terminal and batch set up; and job processing using the various language processors.
- *MUSIC/SP Guide for New Users* (April 1996), introduces new users to the use of MUSIC/SP via an IBM 3270-type workstation. It describes the FSI (Full Screen Interface) menu facility. New users learn how to use many programs on MUSIC/SP for such tasks as editing and running programs.
- *MUSIC/SP Office Applications Guide* (April 1996), describes the features of the TODO (Time, Office, and Documentation Organizer) facility. This includes the scheduling function, spell checking, and MUSIC/SCRIPT (text processing).
- *MUSIC/SP Mail and Conferencing Guide* (April 1996), describes electronic mail on MUSIC/SP. This includes Mail Profile, Mail Directory, using POP clients, and conferencing programs.
- *MUSIC/SP Internet Guide* (April 1996), describes the programs available on MUSIC/SP that provide communication between users through electronic conferencing and discussion lists. Emphasis is placed on access to the Internet with programs such as TELNET (logging on other computers), FTP (File Transfer Protocol), WEB (World-Wide Web), RN (Newsreader), and GOPHER (document search and retrieval protocol).
- *MUSIC/SP Campus-Wide Information Systems* (*CWIS*) *Guide* (April 1996), describes how to create and maintain a Campus-Wide Information System, Help facility, or Classified Ads facility; how to do full-text searching; and how to provide gopher access. MUSIC/SP's resources are used to provide online distribution of information to a wide audience.
- *MUSIC/SP Teacher's Guide* (April 1996), describes various MUSIC/SP facilities related to the academic environment. Emphasis is placed on communication between teacher and student and easy methods for learning how to use MUSIC applications.
- *MUSIC/SP Client/Server* (*MCS*) *Booklet* (April 1996) provides an overview of MCS. Full documentation is available on the MCS diskette.
- *MUSIC/SP Personal Computer Workstation User's Guide* (May 1994), describes the components of the Personal Computer Workstation (PCWS). It is intended for the novice or experienced user of a personal computer, who wishes to connect to MUSIC/SP or another host system. Note that documentation for *PCWS for Windows* is available on the PCWS diskette.

# Trademarks

- MUSIC/SP, TCP3270 are trademarks of McGill University, Montreal, Canada
- IBM, Micro Channel, AS/400, Personal System/2, PS/1, PS/2, and AIX are registered trademarks of International Business Machines Corporation.
- SAA, ES/9370, ES/9000, PROFS, PR/SM, MVS/ESA, VM/ESA, PC-DOS, DisplayWrite, OfficeVision, and GDDM are trademarks of International Business Machines Corporation.
- DEC VT-100 is a registered trademark of Digital Equipment Corporation.
- MINITAB is a registered trademark of MINITAB Inc.
- SAS and SAS/GRAPH are registered trademarks of SAS Institute Inc., Cary, NC, USA.
- UNIX is a trademark of AT&T Bell Laboratories.
- Eudora is a registered trademark of the University of Illinois Board of Trustees, licensed to QUALCOMM Inc. QUALCOMM is a registered trademark and registered service mark of QUALCOMM Inc.
- Other names may be trademarks or registered trademarks of their respective companies.

# Chapter 1. Introduction

# Introduction to Interactive Computing

An **interactive computing system** is a facility through which a number of users can concurrently interact with the resources of one computer. Typically the user is connected to the computer by a device called a *workstation*. A workstation could be a host-dependent terminal or a personal computer. In this guide the term *terminal* is often used to mean the same thing as workstation. Although, technically, a personal computer is not the same as a terminal, it simulates a terminal when it is attached to MUSIC/SP.

This manual describes how to use the MUSIC/SP system. (MUSIC/SP is short for Multi-User System for Interactive Computing/System Product.)

## How to Get MUSIC to Work for You

The first step is to connect to MUSIC from your workstation. This can be done in a variety of ways depending on your type of workstation. Refer to *Chapter 2. Workstations* about how to to connect to MUSIC.

Once you have signed on to MUSIC you can choose one of MUSIC's menu-driven facilities or use MUSIC's command language. Command language is divided into two categories: MUSIC commands and MUSIC Job Control Statements. MUSIC commands tell MUSIC which task you want to do next. For example, you can edit or print a file, invoke a program, or start a menu facility. MUSIC job control statements are included in your files. They tell MUSIC such things as: what computer language your program is written in, where the data is to be read from, and the maximum amount of time for your job. For details refer to *Chapter 5. MUSIC Commands* and *Chapter 6. MUSIC Job Control Statements*.

## Online Information

MUSIC provides online information in the form of electronic bulletin boards, conferencing systems, and help facilities. The MUSIC Help Facility can be invoked by pressing F1 or entering the command HELP. Here, you can find out about all the programs available with MUSIC. Each program you use also provides online help (F1). All MUSIC manuals are online and are available through the text searching facility. Use the command "MAN" to access online manuals.

## Files on MUSIC

The MUSIC system uses two different file structures. One type is called a *Save Library File* and is simply referred to as a "file". This file structure is used for most applications. Each file in your Save Library (directory) can hold up to 56 million characters of data.

The other type of file is called a *User Data Set* or *UDS* file. Although for most applications Save Library files are used, UDS files are of great use in special situations. For example, UDS files can be as large as an entire disk pack if required.

MUSIC has several commands to create, modify, list and purge (delete) files. Also, MUSIC offers a *flat* or *tree* structured file system. With a flat file system, all your files are stored in one directory. It is up to you to devise your own naming conventions through the use of prefixes and suffixes to make distinctions between groups. A tree structured system allows you to make different directories for each group of files. The commands are the same as for DOS (MD, CD, RD). Full details about MUSIC's file system can be found in

*Chapter 4 - File System and I/O Interface*.

## MUSIC/SP Editor

MUSIC has a powerful mechanism for creating and modifying files. It is the *Editor* program, invoked by the MUSIC command EDIT. The Editor has several modes of operation and provides a wide variety of commands for editing files. On most workstations, function (PF or F) keys are used to perform common editing functions. The Editor can search for a particular record based on its contents; perform global changes; move, copy and delete sections; and merge parts of one file to and from another file.

The Editor is fully programmable and can be incorporated into other programs. You can create your own macros and tailor the editor to suit your needs.

## Processors

MUSIC supports many different processors (compilers and interpreters). See *Chapter 8 - Processors* for full details.

Compilers are used to translate programming languages into detailed instructions the computer can work on. The MUSIC system supports the operation of many compilers and others can be added. This manual discusses several different ones. It is possible that your MUSIC system has access to more than those documented here. It is also possible that some of those documented here are not available for your use. Check with your installation about the compilers that are available.

Interpreters such as APL, VS APL, and VS BASIC are similar in concept to compilers, though they are classified as subsystems because they include extensive capabilities beyond that of compiling.

The REXX command executor is documented in Chapter 8. This high-level language allows the execution of MUSIC commands and programs from within the REXX program.

The FSI (Full Screen Interface) menu facility offers many interfaces for processors, eliminating the need to learn job control language. See this topic below for more information.

## System Subroutine Library

A program will often call a subroutine to perform a calculation or get some information. For example, you might want to take the square root of a number, so you call the SQRT subroutine from FORTRAN. Such common subroutines are stored in MUSIC's Subroutine Library.

MUSIC has a great number of additional subroutines available that you can call from your program. For example, you can call one that gives you the current time, another gives the current date, and so forth. *Chapter 9. System Subroutines* contains details of these routines.

## Batch Processing

MUSIC is designed to handle interactive users as well as *batch* users. Batch, with its high speed reader and printer, can be used to run jobs when the interaction from a user is not necessary. The same general command language is used for batch as well as interactive usage.

# Utility Programs

MUSIC includes a collection of programs already written to perform tasks such as sorting and archiving. Most of these programs diagnose errors right away and allow you to correct them immediately. *Chapter 10. Utilities* describes those programs used to perform tasks directly related to MUSIC.

# Menu Facilities

## Overview

Easy access to MUSIC programs can be done through menu (or panel) facilities. This menu approach reduces the need to remember commands and helps novice users to be productive immediately. Menus provide fill-in-the-blank interfaces for many programs to help you specify needed parameters.

MUSIC/SP offers four menu facilities for users and one menu facility for the system administrator. The system administrator facility provides step-by-step menus for installing, monitoring, maintaining, and tailoring the system. It is documented in the *MUSIC/SP Administrator's Guide*.

Each of the four user facilities combine several functions of the MUSIC/SP system. These menu facilities fall under the categories of general users, student computing, teaching applications, and office applications. They are as follows:

- FSI (Full-Screen Interface) - General Users
- CM (Course Management Facility) - Teacher Applications
- CI (Course Information) - Student Applications
- TODO (Time, Office, and Documentation Organizer) - Office Applications

MUSIC/SP provides the programming tools to support the creation and modification of menus. You can easily create your own environment on MUSIC/SP or create an environment tailored for a particular group of users. You can choose to have one of these facilities start automatically each time you sign on to MUSIC/SP. For information about creating and changing menus, refer to the *MUSIC/SP Campus-Wide Information Systems* (*CWIS*) *guide* and the program "TMENU" in *MUSIC/SP Office Applications Guide*.

*Notes:*

1. Your installation may provide additional menu facilities other than the ones described here.

2. As well as using menu facilities on MUSIC, you can use MCS (MUSIC/SP Client/Server) software. Common MUSIC functions (editing files, e-mail, etc) can be done on your personal computer with the use of MCS. For more information use the MUSIC command "HELP MCS".

## Full Screen Interface (FSI)

The Full Screen Interface (FSI) allows you to access various components of the MUSIC system through a series of selection menus. You can specify FSI as the auto-program in your user profile if you wish to have it automatically start when you sign on to MUSIC. This interface is full described, with many examples, in the *MUSIC/SP Guide for New Users*. Help is provided for each screen (program) as it is presented.

To start this interface from command mode (*Go), enter:

```
FSI [item]
```

The *item* is optional and can be one of the highlighted topic names from the main menu of FSI (See Figure 1.1).

Throughout the interface the following standard function key definitions are used.

F1 provides help on the function and usage of the screen currently being viewed.

F3 returns to the previous screen without performing any operation. This can be used to exit from the interface if pressed from the main selection menu.

ENTER performs the operations indicated on the screen.

## Main Selection Screen of FSI

```
Help   End   Up    Down   Top   Bottom  Main   Scan   Find   Topic  Quit
------------------Full Screen Interface for MUSIC------------- Page 1/1
 Command ===>

  Place the cursor on an item and press ENTER or RETURN.

  MUSIC tools:
     Mail          Electronic mail facility
     Programming   Compilers, processors, tutorials, etc
     CI            Course Information
     Internet      Internet access, news reader, gopher, etc
     More          Other general MUSIC tools
  MUSIC files:
     FLIB *        Full Library Screen current directory
     FLIB          Filespec=>                    < pattern
     FUTIL         Other file related utilities
  MUSIC environment:
     Help          General help and online documentation
     New Password  Change your password
     Defaults      FSI customization
     Profile       Profile utility and options
     Disconnect    Terminate your session and disconnect from MUSIC
     /Suggest      Make a suggestion or send a comment to support staff

F1=Help         F3=End           F9=Find            F12=Retrieve
```

*Figure 1.1 - Main Selection Screen of FSI*

# Menu for Teachers (CM)

The Course Management Facility (CM) allows teachers to communicate with their class via the computer. Teachers use the CM command to manage the course material and the students use the CI command (Course Information) to access the information they have prepared.

The Course Management Facility basically does three things. It allows teachers to create and manage files and make them accessible to the class in the form of notes, assignments or a course outline. The files are kept in the teacher's library and are accessible to the students, allowing the students to read them but not change them.

The system also maintains a class mailing list. This list contains the userids and names of the students in the class. It is used to access the student files and to send mail to individuals or the entire class.

Teachers are able to manage the students computer resource allocations and change their passwords.

## CM Main Menu

```
Help   End   Up    Down   Top   Bottom Main   Scan   Find   Topic  Quit
-------------------Course Management for EL101---------------- Page 1/1
Command ===>
     1 *NEW* mail items and    0 replies waiting
 TAB (or use arrow keys) to move cursor to a topic name & press ENTER

 Managing Information:
  MAIL           Receive and send electronic messages
  POP            Update the pop file
  OUTLINE        Update course outline
  NOTES          Update course notes
  ASSIGNMENTS    Assignments management
  LIST           Class list management
  DOCS           Online documentation

 Managing Student Environment:
  TAILOR         Tailor student menu with extra options
  CI             Invoke CI as a student

 Additional Tools:   CONFERENCES   Electronic CONFerences MANagement
                     FILES         View student files
                     AGENDA        Invoke TODO facility
                     TRANS$        Transfer funds and change passwords
F1=Help F2=Ask F3=End F7=Up F8=Dn F9=Fnd F10=Tp F11=Bt F12=Cur PA1=Quit
```

*Figure 1.3 - CM Menu Display*

Full details about this facility can be found in the *MUSIC/SP Teacher's Guide*.

## CI (Course Information)

The Course Information menu is provided for students of teachers using the CM (Course Management) facility. The teacher provides information, such as course notes, assignments, and makes this available through CI. The following menu is a sample that students would work with:

```
Help   End   Up    Down   Top   Bottom Main   Scan   Find   Topic  Quit
-----------------Course Information for CINFO---------------- Page 1/1
Command ===>
 Date: 19May94 14:31:27                          Updated: 10May94 16:30
 TAB (or use arrow keys) to move cursor to a topic name & press ENTER

 General Information:
  MAIL           Receive and send electronic messages
  OUTLINE        Read course outline
  NOTES          The course notes
  ASSIGNMENTS    The assignments
  DOCS           Online documentation
  FSI            The menu interface for MUSIC

  COGN           Conference about Cognitive Computing
  CONFDAT        Conference about Databases
  POLYSOLVE      Calculator
  SN             SchoolNet access

F1=Help F2=Ask F3=End F7=Up F8=Dn F9=Fnd F10=Tp F11=Bt F12=Cur PA1=Quit
```

*Figure 1.4 - CI Menu Display*

Help is provided once the facility is invoked.

# Time, Office, and Documentation Organizer (TODO)

The Time, Office, and Documentation Organizer Facility provides an integrated package tailored to the electronic office environment. This facility is called TODO for short.

To invoke TODO, type TODO from command mode (*Go); the following screen appears:

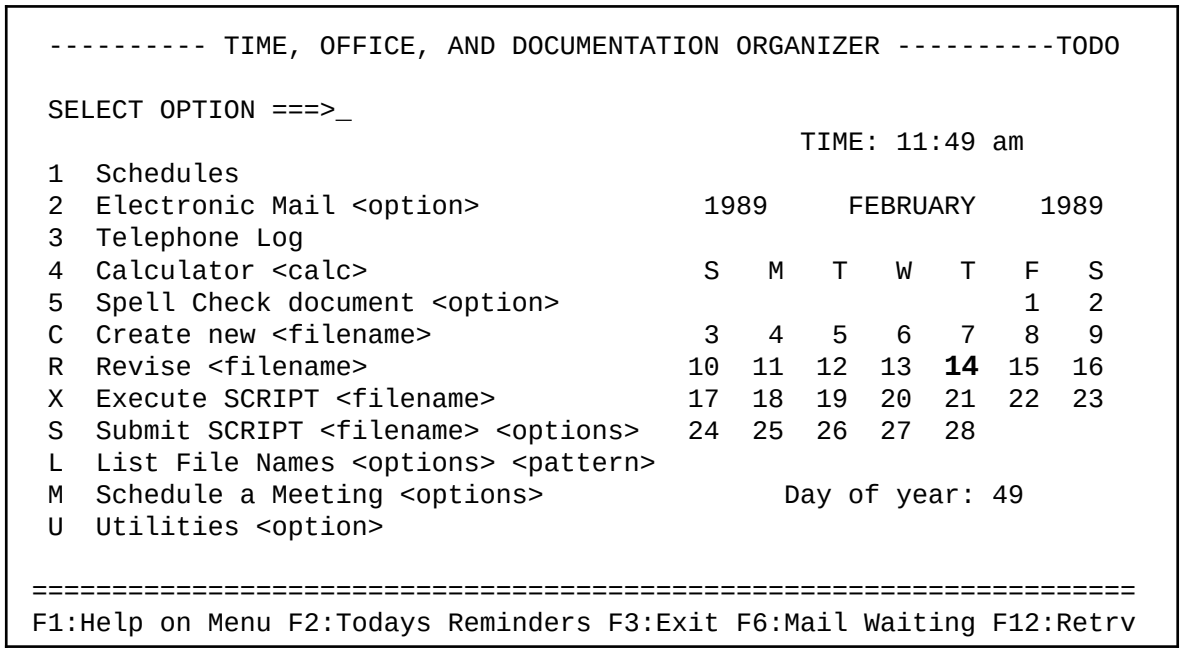



*Figure 1.5 - TODO Menu Display*

The TODO facility allows you to select a function from a menu list. Items include:

- Spelling check of a document or a single word. In document mode, the word and the surrounding lines are presented in an editor-like environment to allow correction.

  Alternative spellings are provided for misspelled words. A 90,000+ word English language dictionary is included.

- A calendaring system allows you to maintain a personal daily schedule. You can authorize others to look at the calendar and allow certain specific users the ability to change items. The system maintains a record of who changed each item.

  This facility can also be used to schedule conference rooms and equipment. Users can directly update the appropriate schedule without the need of sending messages to a room coordinator. Internally the facility uses only standard MUSIC files. A user's appointment data is placed in a separate file for each month to minimize the number of files involved.

- A monthly calendar can be displayed. The user can display future and past months as well as the current one. Reminders can be set keyed on a specific date or all Mondays, etc.

- A log of telephone calls and notes about each call can be maintained.

- An option to assist in the creation of letters and memos according to a user-modifiable style.

Consult the separate publication *MUSIC/SP Office Applications Guide* for full details.

## Word Processing

MUSIC/SP supports three word processing programs. MUSIC/SCRIPT, IBM DisplayWrite/370, and Waterloo SCRIPT (WATCOM Products Inc.). MUSIC/SCRIPT is included with the MUSIC/SP system and is described in detail in *MUSIC/SP Office Applications Guide*.

DisplayWrite/370 is an optional IBM program that may be available at your installation. This program is a host-based text editor and formatter.

Waterloo SCRIPT is a powerful and versatile text formatter, written and distributed by the University of Waterloo. Some of the features of Waterloo SCRIPT include proportional spacing, the ability to generate boxes around text, footnotes, producing multiple columns of text on a single page, creating indexes, and hyphenation. Waterloo SCRIPT supports many popular output devices ranging from simple printing terminals to sophisticated laser printers.

**MUSIC/SCRIPT**

TODO includes MUSIC/SCRIPT as one of its components. MUSIC/SCRIPT is a set of text processing application programs that run on MUSIC.

The MUSIC/SCRIPT facility allows secretarial and administrative, as well as programming personnel to take advantage of the computer's resources for preparing, storing and producing final documents ready for mailing or publication. It is particularly useful for the preparation of letters or technical or legal documents that must be letter perfect or are subject to constant revision.

Corrections and revisions need only be made to the areas actually needing modification, thus saving the need to retype and recheck the unchanged areas. Furthermore, MUSIC/SCRIPT includes features that enable the user to identify those modified areas in the printout of the modified text. The actual modifications to the text are made with the MUSIC editor.

Special control words can be inserted into the input text files to control the format of the output. The output can be immediately displayed at your workstation, or sent to a printer.

The MUSIC/SCRIPT facility includes utility programs which can be of great assistance in the preparation of documents. These include contents and index creation programs.

# MUSIC and You (Userids and Profiles)

## What is a Userid?

When you connect to the system you must identify yourself by entering your userid and password. (Sometimes a userid is referred to as a sign-on code.) Each MUSIC installation may have thousands of authorized userids. Usually each user is assigned a different userid, though some installations may assign a single userid to be used by many users. MUSIC does not restrict the number of users that can be actively signed on with the same userid.

Each userid can be from 1 to 16 characters long and can optionally include a subcode of up to 8 characters. The userid, excluding any subcode, is used to identify the ownership of the files on the system, and is called the *ownership id*. Users can share the same files by having the same userid with different subcodes. Typically, userids do not have subcodes and each person has their own private files.

## Password

Each userid is protected by a password. You must type in the correct password for the userid in order to be allowed to sign on to MUSIC. Passwords can be from 1 to 8 characters in length. It is good practice to use at least 6 characters and not to choose a person's first name as a password, as others may guess it. The password should be kept private between you and MUSIC. It is a good practice to change your password frequently (at least once a month) to protect the integrity of your userid. You can change your own password by using the PROFILE program that is discussed in *Chapter 10. Utilities*.

## Time Limits

Each job that you run on MUSIC is subject to a time limit. This guards against a job going into a *loop* (running indefinitely) for hours when it should have finished in seconds. This limit also protects you from incurring the high usage charge for such jobs.

Time limits are given in *service units*. A service unit is equal to the amount of work done in one second elapsed time on a computer that processes a million instructions per second. (Installations have the option of changing this definition.)

A default time limit is assigned to each userid. This default time will be used if a job does not specifically supply one. You can specify a higher or a lower time limit for a job by way of the /SYS TIME= statement. For information about /SYS see *Chapter 6. MUSIC Job Control Statements*.

## Fund Allocation

The installation may allocate a finite amount of funds to a userid. The amount of funds remaining is updated on a daily basis with processing unit time and connect time charges only. When there are no funds remaining, the user is prevented from signing on until more funds are allocated to that userid. Consult the description of the /ID command and the PROFILE program for ways of finding out the amount of funds remaining in your userid.

# Disk Allocation

The installation may allocate a limit to the amount of disk space that you may use for storing files. A limit can also be set on the maximum size of each file. These limits are applied to individual userids. You can check these limits by using the PRINT command of the PROFILE program. (The PROFILE program is described in *Chapter 10. Utilities*.)

Your disk space allocation limits can be changed by the system administrator.

# Userid Profile

As described above, each userid has a password, default time limits, funds and storage associated with it. Actually, there are many more items for each userid. This list of items is called the *User Profile*. The MUSIC system administrator initially sets up your profile when your userid is authorized. Various limits may be placed on your userid. Naturally, you are not allowed to change some of these limits. There are, however, many items that you can change in your profile. The following list highlights some of the items that are contained in your profile. Consult the writeup on the PROFILE program in *Chapter 10 - Utilities* of this manual for a description of how you can display and modify your own profile.

**Usage Constraints Given in the Profile**

- Time limit for batch jobs
- Time limit for prime-time jobs at your workstation
- Time limit for nonprime-time jobs at your workstation
- Batch access - yes or no
- Maximum User Data Set size that can be created
- Maximum fund allocation for this userid
- Maximum number of extra sessions
- Password changes are or are not allowed
- The ability to save files accessible to others can be allowed or disallowed
- The ability to save files in the common index (public files) can be allowed or disallowed

**User Modifiable Items**

- Userid Sign-on password
- Batch password
- Default printer location
- Whether implied EXEC feature is active or not
- Default job time
- Whether `*In progress` messages are to appear
- Input tab settings
- Output tab settings
- Default tab character on input
- Default backspace character

# MUSIC System Overview

What does the computer do and what does MUSIC do? Even though you can view MUSIC and the computer as one entity, you might find it interesting to find out what MUSIC really is. This section provides a brief explanation of the internals of the MUSIC system. If you are not interested, skip the rest of Chapter 1.

MUSIC can be viewed as just a large program. Actually, MUSIC is one of a class of programs known as operating systems. (Some operating systems are also called *System Control Programs*.) Operating systems are distinguished from ordinary programs in that they can control an entire computer system by themselves and allow other programs to run under their control. Operating systems use special *supervisor* instructions to control the devices connected to the computer. Before describing what the MUSIC operating system does, a brief description of the physical hardware of the computer system is given. (Notice that the word *hardware* is used to describe the physical parts of a computer system while the word *software* is used to describe the programs that run on it.)

## Hardware Components

The heart of the computer system is the Central Processing Unit (CPU). The processing unit gets all its instructions from main storage. Main storage can typically hold several million characters of information and it is generally contained in the same housing as the processing unit.

The processing unit and main storage use external devices to read in and write out information. These Input/Output units (I/O) are connected to the processing unit via channels. There may be many I/O devices sharing and competing for the use of a single channel.

Disk devices are used to extend the storage capacity of the computer system. A single disk can typically store between 300 and 3000 million characters of information depending on the type and model. The information stored on disk is only accessible once it has been read into main storage. The term *blocks* refers to the units of information read in from disk. These blocks are read into areas known as *buffers*. Blocks can also be written from main storage to disk. Buffers can exist almost anywhere in main storage, though on MUSIC most of the buffers are grouped together.

Each workstation is an I/O device, though it may be located thousands of miles away from the processing unit. Each workstation is connected to the channel via a *port* on a transmission control unit. There may be dozens or hundreds of ports available. Each time you use MUSIC you may be connected to a different port, but you needn't be aware of this.

Other I/O devices connected to the channel include magnetic tape and high speed printers, and an operator's console.

## MUSIC System Tasks

MUSIC manages the network of I/O devices, channels, and main storage. It is concerned with space management in main and auxiliary storage. It maintains usage records for accounting purposes. User requests are monitored by MUSIC to protect against violations of system security such as attempts to read another user's private files.

Even though MUSIC is involved in a large number of activities, it still has time to individually respond to

each user's requests and handle them in an appropriate order relative to other users.

## VM and MUSIC

The two operating systems, VM and MUSIC, can run on the same computer at the same time. VM is a special type of operating system designed to run other operating systems under it. With VM, a computer can be used for the MUSIC interactive work and can run the current administrative system at the same time without conversion effort. VM allows the installation to divide the computer's resources, such as the physical I/O devices and the processing time amongst the operating systems under it.

MUSIC running in a VM environment offers some facilities to users which are not available otherwise. For example, TCP/IP services. It is possible to create jobs using the advanced file and editing capabilities of MUSIC and then *submit* them via VM to any other operating system also running under VM. Output from jobs run on many other systems can be retrieved from MUSIC. In this way MUSIC can provide remote job entry (RJE) to the other operating systems that cannot handle workstations directly or that do not handle them in a convenient cost-effective manner. See the SUBMIT writeup in *Chapter 3. Using Batch* for further information.

MUSIC workstations can submit jobs directly to MUSIC's batch facility using VM. This allows jobs requiring disk or tape mounts or those producing large volumes of output, to be scheduled to run directly by users from the workstation. Furthermore, jobs can be submitted to be run overnight, allowing MUSIC to concentrate its efforts during prime time to those problems requiring immediate solution. See the SUBMIT writeup in *Chapter 3. Using Batch* for further information.

## User Region

MUSIC sets aside an area to be used for user programs. In computer terms, one can say the user region is 256K, where *K* represents 1024 characters. (The term *byte* refers to the unit of storage that can hold one character. Typically a computer instruction or a number requires 4 bytes.)

Together with the user's program and data, this user region also contains I/O buffers and control information and a user-system interface module. This interface module is used to communicate requests of the high-level language into requests that are performed by the MUSIC system. This module is also used to present errors to the user in techniques dependent on which high-level language is used. The interface module may also contain some common subroutines used by most programs.

MUSIC supports region sizes larger than 256K. In fact they can be up to several million bytes in size. Check with your installation for the maximum size available to you. If this is too small, use of an *overlay structure* using the *linkage editor,* allows you to run even larger programs.

## User Servicing Techniques

Since many users may want to run programs at the same time, the operating system must have techniques of temporarily stopping one job so as to let another job run. MUSIC uses three techniques to accomplish this: dispatching, time-slicing, and spooling.

The first technique allows another job to run while some other job is waiting for I/O to complete. This is done by a part of MUSIC called the *dispatcher*.

The other technique is called *time-slicing*. A time-slice is typically a small fraction of a second. After each

time-slice, MUSIC checks to see if another user should be serviced next. If so, a region switch is initiated to put the user's job into a dormant state and to activate the other. At a later time, the dormant job can be reactivated to allow the job to continue from where it left off. Active jobs requiring additional storage may cause the system to decide to *swap* a dormant job to disk.

Quite often a job does not go for its full time-slice such as in the following case. A job requires a read operation from a workstation, and it cannot continue until the read operation is done. It may have to wait several seconds or minutes for the user to respond. MUSIC will consider the time-slice is over in such a case, and try to service another user while waiting.

User programs always require the user region while they are running. However, MUSIC can perform many operations without the need for the user region. Examples include the INPUT mode of the Editor on ASCII workstations, the handling of output to the workstation and the initial check of commands which are typed in. MUSIC accomplishes these operations by using a special *spooling* system which allows the user to type in many lines and receive many output lines without the need of the user region. This spool system uses main storage to equalize the demands of the processing unit that can process millions of characters a second to those users who can type about 10 characters per second at best.

Even though the spool system, the dispatcher, and time-slices are fundamental to the operation of MUSIC, the user need not be aware of them, nor do the programs require special instructions to handle them. The beauty of MUSIC lies in the fact that the user need not be concerned with system operation. Application programmers can concentrate on writing and checking their programs; while users of MUSIC subsystems don't even have to know what programs or computers are all about.

# Chapter 2. Workstations

# Overview of Workstations

This chapter describes various types of workstations and how they are connected to MUSIC.

## Basic Concepts

Most people use MUSIC through a typewriter-like device called a workstation or terminal. The term "workstation" and "terminal" are often used interchangeably, although there is a distinction. Workstation is a broader term referring to any type of keyboard device such as a personal computer or a 3270 terminal. Since PCs emulate terminals when they are connected to MUSIC, the word terminal is often used to refer to both types.

Basically, the workstation keyboard is used to transmit commands and data to the computer. After typing text using the alphanumeric keys, you must press the ENTER key to send the data to the computer. The ENTER key informs MUSIC that you have finished typing, and it is only at that point that it will process the data. On some workstations the ENTER key may be referred to as RETURN key.

If you make a typing mistake it can be corrected prior to pressing ENTER by using special local editing keys. Most workstations have a BACKSPACE key which allows you to backup to the error and retype the line from that point. The INSERT and DELETE keys allow you to insert and delete characters in the middle of the line.

A BREAK key is provided to let you interrupt the processing of a command and enter special *break time* commands to inquire about how much time has been used so far, or to skip some lines of output, or to cancel the current job entirely. (3270-type workstations use the PA1 key to provide the BREAK function).

### 3270 Architecture

The MUSIC system runs on computers that use IBM 3270 architecture. This means that any workstation connecting to MUSIC must be able to communicate in this 3270 environment.

There are many types of workstations that can be used on MUSIC. Many workstations emulate the characteristics of an IBM 3270 terminal and are referred to as "3270-type" workstations. This type of workstation is used most often with MUSIC and supports full-screen operation and function keys.

Detailed information on some specific workstations is given later in this chapter.

### Connection to the Computer

All workstations have to be connected to MUSIC via some sort of wire. This connection may be in the form of a regular telephone line (workstation with a modem), or by a direct cable to the computer room.

Workstations which are connected to the computer through a direct cable are likely to be 3270-type terminals. Consult the description of "IBM 3270-Type Terminals" in this chapter for details on making the connection to the system for these types of workstations.

Once the connection to the mainframe is made, you should follow instructions at your workstation to get the MUSIC sign-on screen. If the sign-on messages do not appear, press the ENTER key to clear the screen and the messages should appear. (For specific details about the operation of a particular workstation, consult the

description of the various types of workstations in the later part of this chapter.)

## Types of Connections

The following is an outline of the different types of connections supported by MUSIC. Later in this chapter, each type is covered in its own section to provide more details.

1. **Using 3270-Type Terminals**
   The first major class of workstations is the IBM 3270 family of terminals. They are display terminals that use the EBCDIC encoding conventions. They connect via a coax line to a 3270 controller. Since these terminals are usually directly connected they offer a good data transfer rate. A number of different terminal models are part of the 3270 family. The IBM 3270/PC is a special one in that it can also operate as a stand-alone PC.

   Most IBM host software is written to support 3270-type terminals. These operate in Full Screen mode, providing programs such as the MUSIC Editor, with the ability to define input and output areas anywhere on the screen.

   *Note:* Many types of workstations can emulated 3270s and are referred to 3270-Type terminals. Each of the other types of connections below provide some method of emulating 3270s - thereby, taking advantage of full-screen mode of operation.

2. **Using ASCII Terminals**
   The second major class consists of ASCII workstations. These can connect directly to the computer or by using a modem. Depending on the computer hardware at your installation, your workstation may operate in line mode or, if your site has a protocol converter, it can emulate a 3270 terminal. Your installation should be consulted for these details.

3. **Using a Personal Computer with PCWS (or other communications package)**
   PCWS is a terminal emulation program distributed with MUSIC and designed to run on computers from the IBM PC, PS/1 and PS/2 families. This communications package allows your PC to be remotely connected using a modem. Designed specifically for communicating with MUSIC, it provides reliable file transfer and 3270 terminal emulation without a protocol converter.

   PCWS also provides VT100 terminal support if your PC is connected via a protocol converter. The VT100 terminal type is widely used with many types of computer architecture.

   Although using PCWS to communicate with MUSIC offers a number of advantages, PCs can also use other commercially available communications programs.

4. **Using NET3270 for Personal Computers on a LAN**
   NET3270 is an optional product that can be purchased through the MUSIC Product Group. This software provides 3270 emulation to all IBM PC, PS/1 or PS/2 connected to a LAN (Local Area Network). The LAN hardware can be Token-Ring, Ethernet, or Arcnet.

5. **Using the Internet**
   If your workstation is running TCP/IP software and has access to the Internet, you can sign-on to a MUSIC system. You can connect from your workstation or from another computer system. The command you use depends on the software - the most common commands are "TELNET" and "TN3270".

More details about each type of connection is given later in this chapter. This information serves only as a guide for operating the workstation and should not be used in deciding which workstation to get. Your installation can usually be of assistance in advising you about this matter. You should now read that section

pertaining to the type of workstation you intend to use. Alternately, you could try to find someone who would take a few minutes to explain the workstation operation to you.

## Workstation Defaults

When you sign on, the system can automatically distinguish between 3270 and ASCII workstations and it will select the appropriate operating characteristics accordingly. Since not all workstations within each of classes are the same, sometimes additional information must be supplied indicating the exact model of workstation being used. This information can be specified using the *trmcls* option of the /ID command or the TERM option in your profile. Other profile options can also be used to set personal defaults for such things as input and output tab settings. These settings can also be dynamically changed by MUSIC commands during the time you are signed on to MUSIC. Refer to the writeup on the PROFILE program in *Chapter 10. Utilities* section of this manual for more information about how you can examine and change the profile defaults.

# IBM 3270-type Terminals (Connection type 1)

## Supported Models

MUSIC supports the 3178, 3179, 3180, 3277, 3278, and 3279 display terminals (80 and 132 characters per line models) of the IBM 3270 family when they are locally attached to the processing unit. It also supports the 3270 Personal Computer, Personal Computers with 3270 co-axial adapters, and terminals connected through 3270 protocol converters such as the 7171.

Remotely connected terminals can be supported to appear as though they are locally attached terminals to MUSIC. The software and hardware to do this is outside of the domain of MUSIC.

## Special Keys

The ENTER key is used on the 3270-type terminal as an end of line signal. It is located on the bottom right of the main keyboard. The BACKSPACE key is used to correct typing errors you detect in a line before the ENTER key is pressed. There is no limit to the number of times you use the BACKSPACE key.

On 3178, 3180, 3278 or 3279 terminals, the ALT key to the right of the space bar has special significance. On some terminals, it must be held down while the PA1, PA2 or CLEAR keys are pressed.

The PA1 key is used to generate the break signal, which puts the terminal into attention mode, indicated by `**Attn**` in the lower right corner of the screen.

The PA2 key is used to clear the top portion of the screen. This key can be pressed in response to a `More...` status. With MUSIC, the PA2 key is also used during full-screen programs to request a Multi-Session function. Multi-Session is described later in this chapter. (On some keyboard arrangements, this key may be marked CNCL instead of PA2.)

The PA1 and PA2 keys on some keyboards perform a dual function depending on whether the SHIFT key is also pressed. For example, the PA2 key may have FIELD MARK also written on it. If this is so, then pressing this key may not result in the display clear operation you think it should. This could be the result of pressing the LOCK key accidentally causing the keyboard to be locked in upper case. Pressing the SHIFT key solves this problem.

The DUP, TEST, and FIELD MARK functions are generally not used on MUSIC.

MUSIC supports the APL ON/OFF key. Refer to the VS APL description for details on this usage.

## Entry Assist Feature

3178, 3278 and 3279 terminals can have an entry assist capability available on them. This assist provides many desirable features such as automatic wordwrap, column tabbing, end-of-line signal and cursor positioning information. The feature is enabled by pressing the DOC ON OFF key located below the key marked ATTN. Consult your installation for information on how to use this feature.

# Power On & Display Intensity

## 3277

Turn the terminal on by gently pulling the power-on knob located at the bottom left of the screen. You can rotate the front of this knob for the proper intensity. It is best not to have the intensity too high. The intensity of the characters that you type in can be made less than the rest of the display. This is controlled by the back half of this knob.

## 3178/3278/3279

Turn the terminal on by pushing the red switch located at the left of the screen. The two dials at the right of the screen control the intensity and contrast of the displayed characters. The two switches on the right of the screen should be set in their "A,a" and "Normal" positions.

# Sign on to MUSIC

Once the terminal is powered on it will usually display the word MUSIC/SP in block letters. Press the ENTER key to get the sign-on message.

If the terminal displays the characters VM then you must press the ENTER, type "DIAL MUSIC" followed by the ENTER or PA2 key.

If neither MUSIC or VM is displayed, press the CLEAR key.

*Note:* For purposes of the Editor, it is important for MUSIC to know the actual number of PF Keys (program function keys) on the terminal being used. At sign-on time, if the /ID line is entered by pressing the ENTER key, MUSIC assumes 24 PF Keys if the terminal appears as an IBM 3178, 3278 or 3279. Otherwise (e.g., IBM 3277), MUSIC assumes 12 PF Keys. The user can override these assumptions by pressing the highest numbered PF Key to enter the /ID line at sign-on time, rather than the ENTER key. The highest numbered PF Key is PF12 or PF24 and is always located at the extreme lower right hand corner of the keyboard. Refer to the description of the /ID command in *Chapter 5. MUSIC Commands* or to the "Editor Full-Screen Mode" description in *Chapter 7. Using the Editor* for more information.

# Screen Format

MUSIC uses the top lines of the screen as an output area. You normally type information in an input area located one line from the bottom of the screen. A line formed of dashes and the letter T will appear directly above this area. The location of the T's correspond to your TAB settings and you can change their location by the /TABIN command. The TAB key on the 3270 is not normally used for any tabbing effect.

A MUSIC state message appears at the bottom right of the screen, for example, "Reading".

A special cursor character that looks like an underscore (_) shows the location where your next typed character will appear.

## Screen Control

The display screen output area fills up starting with the top line of the screen and continues down until all the lines in the output area are filled. Lines greater than 80 characters will appear on two lines. The `More...` message will appear when MUSIC wants to display a line and no more space is left in the display area. Press the ENTER key to clear the display area and allow MUSIC to continue its output. Some programs may issue commands to start at the top of the screen without completely filling the display area. The `More...` message will be displayed in this case as well.

## MUSIC Screen States

The MUSIC screen state message at the lower right can be one of the following:

| | |
|---|---|
| Reading | MUSIC is waiting for the user to enter some information. |
| **Attn** | The terminal is in *BREAK* status waiting for the user to type some command. This status is caused by pressing the PA1 key. |
| Working | MUSIC is working on your last request and no output is currently available. |
| More... | MUSIC is waiting for the user to press the ENTER key to clear the output display area, since it has more output waiting. |
| Writing | MUSIC is currently writing information on your screen. You may never see this message as MUSIC can normally write to the screen faster than your eyes can detect. This message will, however, appear when the system is processing your /CANCEL request. |

## Advanced Typing Techniques

This terminal allows for advanced character editing of input information before it is passed to MUSIC. You will notice that the BACKSPACE key immediately corrects any typing error.

You are not able to change anything outside your 80 character input area. Attempts to do so will cause the INPUT INHIBITED screen status indicator to come on. The RESET key can be used to turn this indicator off. Then, move your cursor back into the input area with an arrow key.

The cluster of 4 keys with arrows pointing in different directions can be used to move the cursor anywhere on the screen. The DEL key can be used to delete the character in your input area pointed to by the cursor and results in the rest of the line being shifted one character to the left.

The INS MODE key can be used to insert some characters at the point identified by the cursor. You must press the RESET key when you are finished the insert operation.

Pointing the cursor to any line in the output area (including the TAB line), followed by pressing the ENTER key, will cause the line to be copied to your input area. This is called *line call down*. Once in your input area, you can modify it by using the character editing facilities of the terminal. Press ENTER to accept this line as input. (Note that this *line call down* feature copies the line as **displayed.** Thus if the original line had some unprintable characters in it, they will not appear in the copy brought down to your input area.) Line call-down cannot be used on 3270s connected via VM Remote Dial.

The RETRIEVE function (PF12) retrieves the last command entered and scans back through these

commands each time PF12 is pressed. These commands are displayed in the input area and can be modified and re-entered. (See the /DEFINE command in *Chapter 5. MUSIC Commands* for additional information.

The ERASE INPUT key can be used to completely clear the input area. The ERASE EOF will clear the input area from the cursor position on to the end of the input area.

MUSIC allows the user to start typing the next line of information before it has finished processing the first. Thus while your terminal is in Working or More status, you can type in the next line. You can even press the ENTER key if you are sure you want this line next.

## Notes

1. A BREAK request can be signalled when the terminal is in More status by pressing the PA1 key. This will first clear the screen and then put the terminal in ATTN (break) mode.

   While the terminal is in attention mode (ATTN), you can press PA1 to skip the remaining output. This is equivalent to entering the command /SKIP ALL. Thus, to skip the remaining output when in More status, press PA1 twice (the first one enters ATTN mode and the second one does the skip).

2. The space bar or the special cursor movement keys can be used to move the cursor to the right. They are not equivalent. The cursor movement key does not put in blanks as it moves. Consequently if you extend your input line by using the cursor keys you will find that the apparent blank characters are not really there when you press the ENTER key.

3. All lower case characters will be displayed as their upper case equivalents on some 3270 terminals, but the terminals still send MUSIC the lower case characters. This presents no problem to the user unless the user is working with a file that was created with TEXT LC or equivalent. Such is the case when the file is to be used by the MUSIC/SCRIPT subsystem.

4. Pressing the CLEAR key when you turn on your terminal and the previous user has not signed off, enables MUSIC to initialize your display. If your display is not initialized in this manner, the output will not be formatted correctly.

## 3270/PC and other Co-Axial Connected PCs

MUSIC supports PCs connected via co-axial cable. The IBM 3270 Personal Computer uses that kind of connection. PCs connected in this way essentially function as regular 3270 terminals. The file transfer can be provided using the IBM TSO file transfer program. The procedure for transferring a file is as follows:

### SEND : Transferring a file from the PC to MUSIC.

1. Sign on to your MUSIC system from a host session in 3270 mode.

2. Make sure the screen is cleared. You can either press the CLEAR or PA2 key to do this. Failure to do this will cause an error message to appear. Follow instructions at the end to recover from this situation.

3. When running on a 3270/PC you must press the JUMP key located on the left hand side of the keyboard to go to the PC session. If you have windows defined, the window must be expanded to full size.

4. Issue the following command when the DOS prompt appears:

```
SEND fromname x:toname  [ASCII] [CRLF] [APPEND] [RECFM(V|VC|F|FC)]
                        [LRECL(n)] [SPACE(n)]
```

where:

| | |
|---|---|
| fromname | is the name of the file being sent to MUSIC. This name can include a drive specification and a path. (Refer to *Specifying the Path to a File* in your IBM Personal Computer DOS manual for further information.) This is a required option. |
| toname | is the name of the MUSIC file where the transferred file will be saved. This is a required option. |
| ASCII | specifies that the file stored on the PC is stored in ASCII format. It has to be converted to EBCDIC during the transfer to MUSIC. When sending text files, you should always specify the ASCII parameter. |
| CRLF | specifies whether carriage return/line feed characters are recognized as record separators and deleted before storing the data onto MUSIC. |
| APPEND | Allows you to attach the PC file to the end of a MUSIC file. For a very large file, you can pre-allocate the file on MUSIC (use (NORLSE,RECFM(VC)), then use the APPEND option. |
| RECFM | specifies the record format of either V, VC, F, or FC. |
| LRECL | specifies the logical record length. **n** can be from 1 to 32xxx. |
| SPACE | specifies the space needed up to 4000K. |
| CUT | This option must be specified if your terminal is operating in CUT mode. DFT mode is the default and needs no specification. |

## RECEIVE : Transferring a file from MUSIC to the PC.

1. Sign on to your MUSIC system from a host session in 3270 mode.

2. Make sure the screen is cleared. You can either press the CLEAR or PA2 key to do this. Failure to do this will cause an error message to appear. Follow instructions at the end to recover from this situation.

3. Press the JUMP key located on the left hand side of the keyboard to go to the PC session. If you have windows defined, the window must be expanded to full size.

4. Issue the following command when the DOS prompt appears:

```
   RECEIVE toname   fromname  [ASCII ]  [CRLF ]  [APPEND ]
                  B:fromname
```

where:

| | |
|---|---|
| toname | is the name of the PC file where the transferred file will be saved. This name can include a drive specification and a path. (Refer to "Specifying the Path to a File" in your IBM Personal Computer DOS manual for further information.) This is a required option. |
| fromname | is the name of the file being sent from MUSIC. This is a required option. The "B:" is needed for PC/3270 session ID. |
| ASCII | specifies that the data should be stored in ASCII format on the PC. A translation from EBCDIC to ASCII will be done. |
| CRLF | specified whether carriage return/line feed characters should be inserted as the last two characters in each line when a MUSIC file is stored on the PC. |
| APPEND | Allows you to attach a MUSIC file to the end of a PC file. |

## Notes

1. Failure to have cleared the screen prior to starting the file transfer will cause the following error message to appear on the PC session.

   ```
   TRANS08 Command Transfer Error: File Transfer Cancelled.
   ```

   To recover from this condition you must:

   a. Go to the host session. You should see the message `More...` in the lower right hand corner of the screen.

   b. Press ENTER. The screen will clear and the cursor will move to the lower right hand portion of the screen.

   c. Press the PA1 key THREE times.

   d. Press the ENTER key. The file transfer program (on the host) will be terminated. The PC side of the file transfer program was already cancelled.

   You may then restart the file transfer request.

2. Refer to the *IBM 3270 Personal Computer Control Program User's Guide and Reference* (SC23-0102) for more information on using the 3270/PC and file transfer using the 3270/PC.

# ASCII Terminals (Connection type 2)

## Terminology

MUSIC supports the Teletype Models 33, 35, 38 and 43. (*Teletype* is a trademark of Teletype Corporation, Skokie Illinois.) Other models may also work. Some other manufacturers produce terminals which may work on MUSIC in a similar fashion to those made by Teletype Corporation. The term *TTY* is used as a generic name referring to this general class of ASCII terminals. In many cases the operation of ASCII terminals on MUSIC is independent of its manufacturer.

MUSIC supports the IBM 3101 ASCII display terminal in character mode. This terminal is one of the many that form the TTY class of terminals supported by MUSIC. The IBM publication *IBM 3101 Display Terminal Description* (GA18-2033) contains full details of the keys, switches and operation of this device.

MUSIC supports the IBM 3161, 3163 and 3164 ASCII display terminals in either native or 3101 mode. You can obtain a complete description of the setup, operation, and capabilities of these terminals in the appropriate IBM publications.

## Trmcls

The correct specification of the *trmcls* parameter on the /ID command is quite important for the correct operation of some ASCII terminals. Consult the writeup under the /ID command in *Chapter 5 - MUSIC Commands* for more details.

For example, on an IBM 3101, the sign-on command would be "/ID userid;3101".

On an IBM 3161, the sign-on command would be "/ID userid;3161".

## Special Keys

Most installations allow users to use the RETURN key as the end of line signal. (If your installation does not, then you must generate this signal by pressing the Q or S key at the same time as you are holding down the CTRL key. You will find it easiest to press the CTRL key first and while keeping it down, press the S or Q key.)

Some models have a separate BACKSPACE key. On others, the backspace function is accomplished by typing the letter "O" at the same time as holding down the SHIFT key. A left pointing arrow (←) or an underscore (_) character will usually print when this is done. The carriage may not actually backspace, but the equivalent function is carried out by the computer.

The BREAK key is used to perform the break function. It can also be used to cancel an entire line, prior to pressing the RETURN key.

The REPT key can be used to ease the typing of the same character many times. Just press the desired key at the same time that you are holding down the REPT key. Release the keys to stop the repeat action.

The upward pointing arrow character is interpreted exactly the same as the vertical bar (|) character on other terminal types.

# Terminal Set Up and Connection

Some terminals can operate at different speeds depending on a switch setting. Sometimes the speed switch is marked in characters per second (cps) which is roughly 1/10 the baud rate.

If the terminal has a switch marked full duplex (FDX) or half duplex (HDX), choose half duplex. If the coupler has a half/full duplex switch, choose full duplex.

Establish the connection to the computer. MUSIC may immediately type a message asking you to sign on. If no message appears, press the RETURN key which will then cause the message to be displayed.

You should now sign on without undue delay. Waiting longer than twenty-five seconds to sign on may cause MUSIC to disconnect you.

# Special Features and Considerations

TAB characters may be sent to the computer by pressing the letter I key while holding down the CTRL button. The carriage on your terminal may not actually move but the computer will take the proper action.

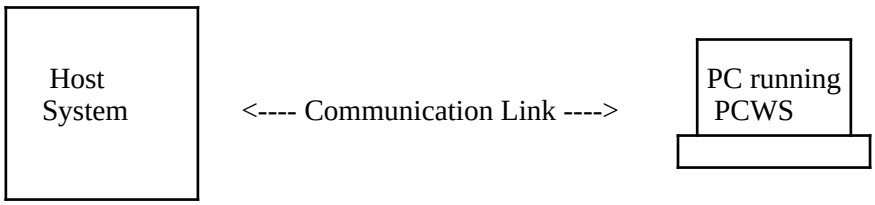
# Controlled Scrolling for ASCII Video Terminals

Scrolling is a function of video display terminals by which, as new lines are added at the bottom of the screen, old lines automatically disappear (scroll) off the top. When doing output to a terminal of this type, the situation where the output scrolls off the top of the screen before you have a chance to read it can occur. This problem can be avoided using the "/CTL LINES=nnn" command. (*nnn* specifies the number of lines on the video display screen).

When the screen fills up, MUSIC will ring the bell at the terminal and pause. To continue displaying output, press the RETURN key (or equivalent key). Alternately, your installation may have set up a special terminal class to be used with the type of terminal you are using. (See the trmcls parameter of the /ID command). In this case when the screen fills up the message `More...` appears in the lower right hand corner of the screen. The RETURN key can be pressed to continue when convenient. On terminals connected over packet switched networks, you may have to press the BREAK key instead of the RETURN key to continue output.

# PCWS (Connection type 3)

MUSIC's PCWS (Personal Computer WorkStation) is a Terminal Emulator Program designed to run under the DOS operating system on computers of the IBM PC and PS/2 families. PCWS allows a PC to communicate with the outside world through one of its serial ports.




Starting with PCWS version 2.30, the file PCWS.EXE contains the executable program. (Earlier versions of PCWS consisted of two parts: CLM.COM and TERM.COM.)

For more detailed information, please refer to the *MUSIC/SP Personal Computer Workstation User's Guide*.

## PCWS Terminal Types

The PCWS program makes it possible for a PC to emulate one of the following two terminal types:

### PCWS Terminal Type

The PCWS terminal type is a special one, designed specifically for communicating over an asynchronous line with the MUSIC operating system. This terminal type has two different modes of operation:

1. PAGE Mode
2. 3270 Mode

The initial mode of the PCWS terminal type is known as the PAGE mode. In this mode, the PC basically emulates a standard line-by-line ASCII terminal. PAGE mode remains the normal mode of operation until a MUSIC full-screen application is started. The PCWS terminal type then automatically switches into 3270 mode. This second mode allows full-screen applications (such as the Editor Mail, etc.) to run on the PC as if it was a real 3270 terminal. Upon terminating the 3270 application, PCWS automatically switches back into PAGE mode.

### VT100 Terminal Type

The VT100 terminal type of PCWS allows a PC to emulate the DEC (Digital Equipment Corporation) VT100 terminal. This VT100 emulation capability enables a PC to be connected via a protocol converter.

A protocol converter is a device allowing ASCII terminals to connect to a host computer as if they were IBM 3270 terminals. A PC using the VT100 terminal type of PCWS appears to the host as a regular 3270 terminal. The converter's role is to translate the 3270 data streams it receives from the host into ASCII control sequences the terminal recognizes, and vice versa. As for PCWS, it makes sure the PC processes incoming and outgoing data just as a real VT100 terminal would.

A protocol converter holds a 'Device Definition Table' (DDT) for every ASCII terminal it supports (including VT100). Converters such as the 7171 and the 9370 ASCII Subsystem are shipped with default DDTs for a number of terminals. Each of these supplied DDTs allows a different terminal type to be connected to the converter. During the connection procedure, the protocol converter must find out which DDT to use. To do so, it issues the following prompt:

```
"ENTER TERMINAL TYPE:"
```

A special DDT called 'VT100P' is distributed with MUSIC and should be installed on your site's converter by a system's administrator. The entries of this table provide enhanced key definitions and screen colors.

## Other PCWS Features

PCWS was designed specifically to exploit the level of interfacing between MUSIC and the PC. As an example, file transfer is initiated by entering the XTMUS or XTPC command from the MUSIC session. The PCEXEC command of MUSIC executes PC applications as if they were entered at the DOS prompt. Also, many features included in the MUSIC Mail program are facilitated when connected with PCWS.

## PCWS for Windows

The original PCWS is designed to run under the DOS operating system. Another program called PCWS for Windows also exists which comes with its own Windows formatted help files.

# NET3270 for Personal Computers (Connection type 4)

The Network 3270 Workstation Program provides 3270 terminal emulation to users on local area networks. The NET3270 program also provides multiple session services and file transfer capabilities with the following IBM host operation systems: MUSIC/SP, MVS/TSO, and VM/CMS. NET3270 currently provides 3270 Model 2,3, and 4 terminal emulation which includes extended data stream support, file transfer, HLLAPI (High Level Language Application Program Interface), and support for IBM's GDDM-PCLK Version 1.1.

NET3270 uses either the NETBIOS Session Services transport protocol or Novell's SPX (Sequence Packet Exchange) protocol to provide a communications link between client workstations and NET3270's dedicated communication server.

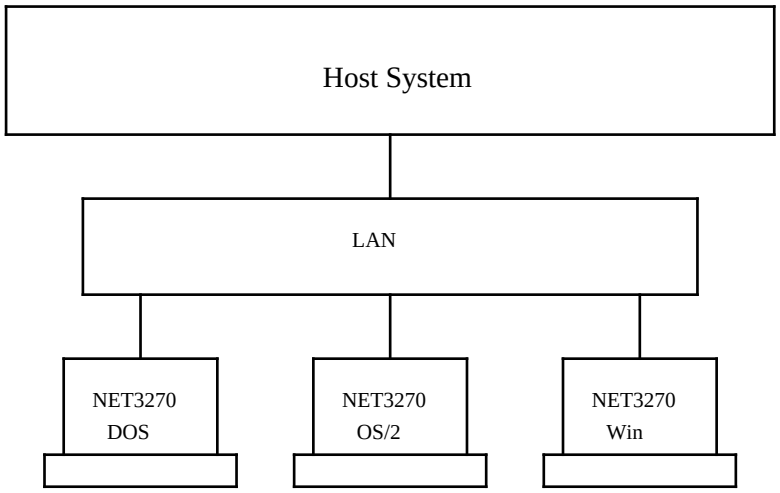



*Figure 2.1 - Sample NET3270 Configuration*

## Starting NET3270

Assuming that NET3270 is installed on your PC and that you are connected to your LAN network, enter "net3270 -r" from your DOS prompt. For example,

```
F:> net3270 -r
```

This will load NET3270 in resident mode (described below). Omit the "-r" if you wish non-resident mode.

## NET3270 Features

NET3270 has many features including file transfer and multiple sessions. If you use NET3270 in resident mode you do not need to logout or exit a host application in order to do some work on your PC. You can use the F1 hotkey to toggle back and forth between the host session and the PC session.

The following help text gives you a quick glance at the features of NET3270.

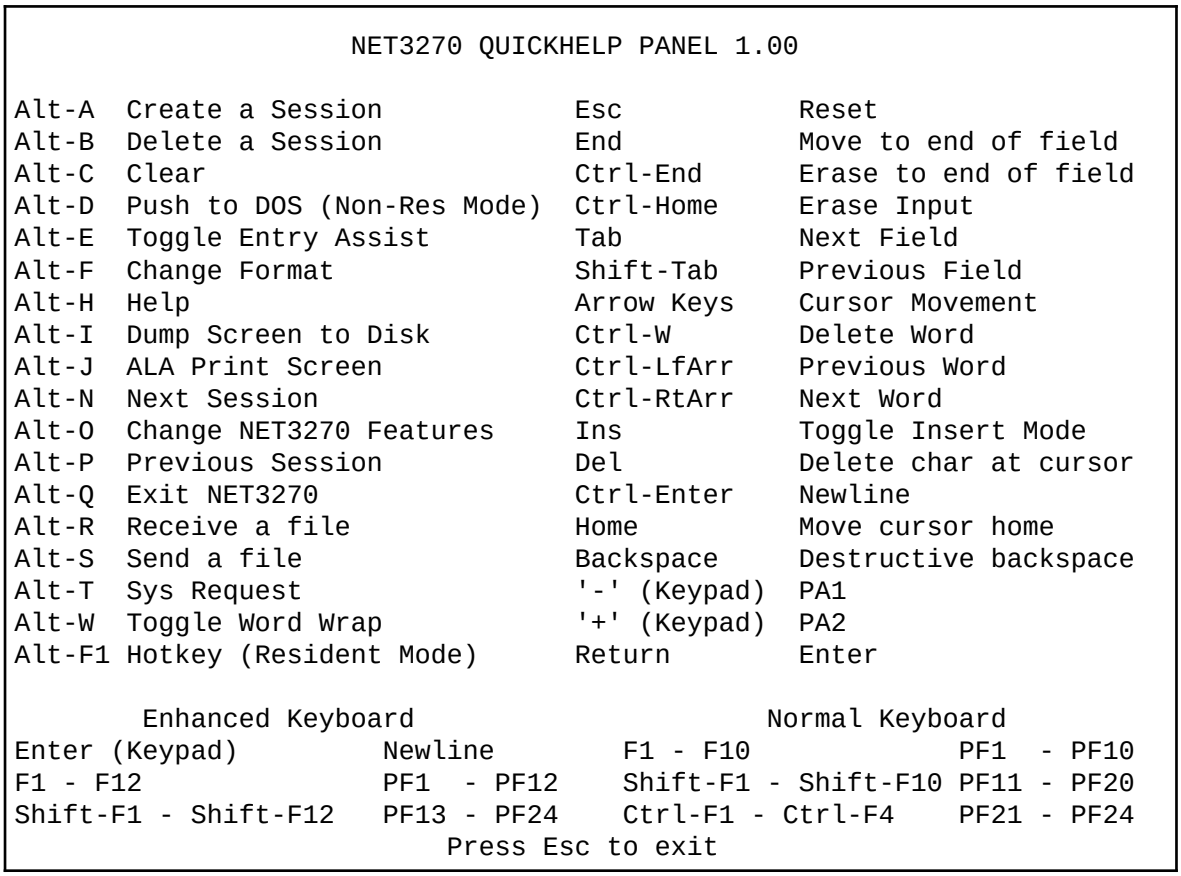



*Figure 2.2 - NET3270 Help Screen*

For further information, refer to the following publications:

*NET3270 - The Key to Connectivity - Workstation User's Guide*
*NET3270 - The Key to Connectivity - Administrator's Guide*
*NET3270 - The Key to Connectivity - Programmer's Guide*

# Internet Access (connection type 5)

## Using the Internet to Connect to MUSIC

Since the 70's the Internet has evolved to encompass a large number of heterogeneous sites world-wide, and allows you to reach around the world and contact other computer sites. Your computer and the host must be connected via TCP/IP to the Internet in order for you to communicate.

The command you use to access systems with 3270 data streams (like MUSIC and VM) depends on the TCP/IP software that you are using. Two common commands are "TELNET" and "TN3270".

### Example of Accessing a MUSIC System

McGill University allows guests to access its CWIS (Campus-Wide Information System) called "infoMc-Gill". (To use other facilities on MUSIC you need a userid and a password.) InfoMcGill provides the phone directory, library information, positions available, etc. The following example includes McGill University's Internet address.

```
TELNET vm1.mcgill.ca            or        TN3270 vm1.mcgill.ca
```

Once connected, choose "infoMcGill" after the VM logo.

## MUSIC's TELNET Command

You can use the TELNET command of MUSIC to connect to any machine on the Internet.

Each site that you visit can be a new system to learn. Follow the instructions carefully. (MUSIC's TELNET has help (F1) when it is first invoked - or type "HELP TELNET" at the *Go prompt.) For more information about TELNET and other Internet commands see the *MUSIC/SP Internet Guide*.

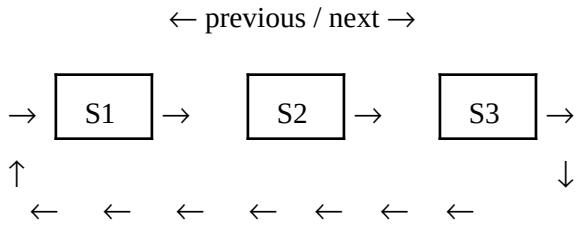
# Multi-Session Support

## Overview of Multi-Session

Multi-Session Support allows a number of separate sessions to be signed on to MUSIC from one terminal. This support provides the 3270 user with the ability to interrupt one task to perform another. For example, while responding to electronic mail, you realize that your response depends on some information stored in a file. You can suspend the first session and create a new one by pressing a function key. After locating the information you need, press another function key to return to the previous session and resume answering your mail. Another example of using Multi-Session Support involves using a second session while the first session is executing a program.

At any instance only one session is actually in control of the terminal. This is known as the active session.

Function keys are used to create, delete, and switch between sessions. The following diagram represents three sessions.




When you first sign on, one session is allocated. As you create subsequent sessions, they are connected to the original as illustrated above. The functions NEXT (F8) and PREVIOUS (F7) are used to switch active sessions. If S2 is active, then NEXT activates S3, whereas PREVIOUS activates S1. Note that if S3 is the active session NEXT makes S1 active.

The ADD function (F4) creates a new session and connects it after the current active session. The newly created session then becomes the active session. For example, if the ADD function is invoked from S2 then a new session is added between S2 and S3. The number of sessions that a user can create is limited by the number allowed in the user profile and the number of multi-sessions allowed system wide. The ADD function may fail with an error message if either of these limits are exceeded.

The DELETE function (F5) removes the active session from the chain and activates the session preceding it. If S2 is deleted then S1 is the new active session. When a session switch occurs, the system displays the session ID number of the new active session in the message area of the screen. This ID number is the same Terminal number (Tnum) which is displayed by the /STATUS command.

## Usage of Multi-Sessions

Multi-Session Support is only available on 3270 terminals and terminals which emulate 3270's. The functions described above are selected by pressing the appropriate program function key. The default definitions of the program function keys are listed below. The /DEFINE command can be used to change these defaults. (See /DEFINE in *Chapter 5. MUSIC Commands* for more information.) To display your current function key definitions enter SHOWPFK in *Go mode.

| | |
|---|---|
| F4: ADD | A new session is added between the active session and the next one in the chain. |
| F5: DELETE | The active session is deleted and the previous session becomes the new active session. This is only valid if the active session is in *Go mode (not running any program) and it is not the only session left. The OFF command has the same effect as F5 except it also works for the last session. |
| F7: PREVIOUS | Make the previous session in the chain the active session. |
| F8: NEXT | Make the next session in the chain the active session. |
| F9: PRINT SCREEN | Prints the current screen image on a printer. F9 invokes the REXX procedure called PRTSCR. This procedure copies the current screen image to a file called @PRT and issues the PRINT command. You can use the ROUTE command to change the default printer before pressing F9. Also, you can create your own "PRTSCR" and have the screen image appended to a file instead of printing it. |

Programs that use the full screen interface, such as the EDITOR, TODO, and MAIL, have the program function key information passed directly to them, bypassing Multi-Session Support. To inform the system that the program function key is a multi-session request, the user must press the PA2 key before pressing the function key. If the PA2 key is accidentally pressed, the ENTER key can be used to cancel the multi-session request. The following summarizes the sequences required from *full-screen* programs.

PA2, F4 - ADD
PA2, F5 - DELETE
PA2, F7 - PREVIOUS
PA2, F8 - NEXT
PA2, F9 - PRINT SCREEN

The above functions have equivalent MUSIC commands. You can process MUSIC commands by entering them in the command area of a full screen program. The command must be preceded with a slash (/) to inform the current program that you are entering a MUSIC command. For example, /ADD to add a new session. (/DELETE is only valid in *Go mode.)

*Notes:*

1. Programs continue to run in non-active sessions unless one of the following events occurs: 1) the program requests input from the terminal; 2) the program's output exhausts the supply of output buffers. When one of the above events occurs, the program stops execution until the session is re-activated.

2. From an accounting perspective, using another session is the same as signing on to another terminal. Therefore, users should realize that multiple sessions are charged accordingly.

3. While using the Editor program, it is important to note that changes to the current screen are not recorded when you press PA2. Make sure that you have pressed an action key (i.e. ENTER) before switching sessions.

4. The program named MS can be executed in *Go mode to obtain information about your current sessions. The session ID, userid, and the last command entered are listed for each session starting with the active one.

# SESSIONS Command

The SESSIONS command is used to show information about multiple sessions signed on to your userid. The following is an example:

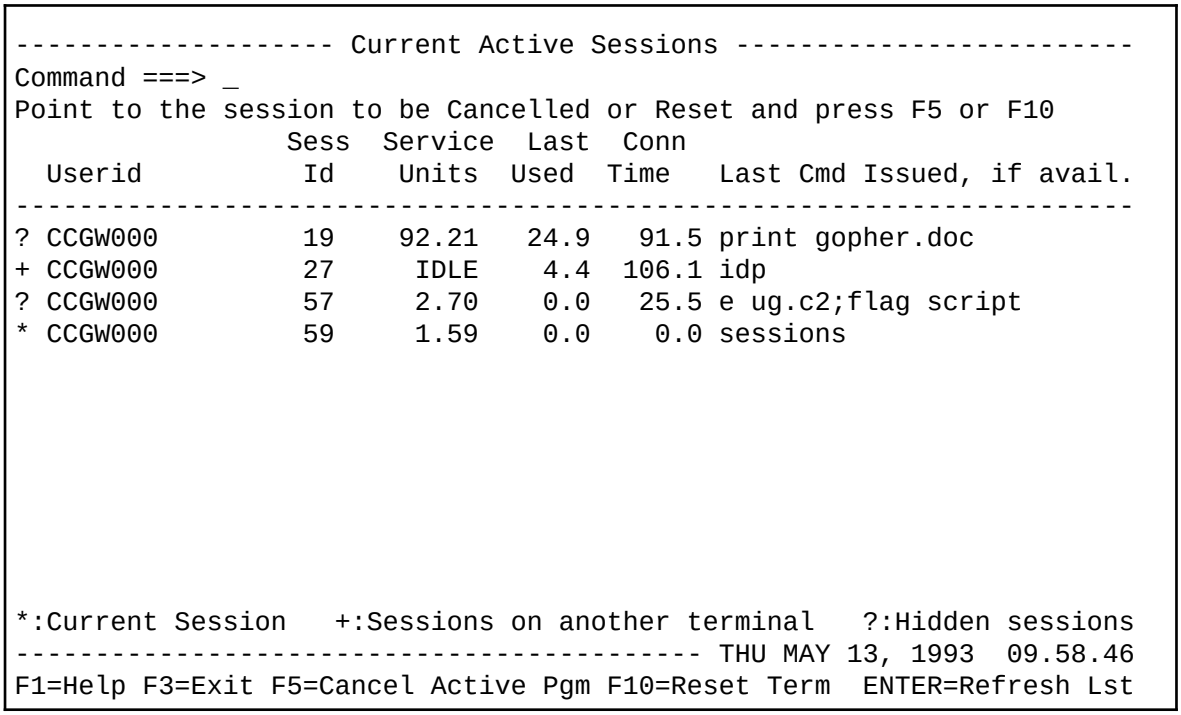



*Figure 2.3 - Screen display for SESSIONS command*

# Workstation Output Control

## Overview of Output Control

The following section contains material that can help speed up the display of information on slow-speed terminals. With the exception of the /CANCEL and /SKIP commands, you may wish to skip this section as the other features described may not pertain to your workstation.

MUSIC has many features that enable the user or the program to control the input and output to the workstation. MUSIC allows you to dynamically skip output, to compress out multiple blanks, or to just display a piece of each output line. The user may also immediately stop the output, enquire how much job time has been used so far, etc.

## Dynamic Output Control

MUSIC allows you to interrupt the output by pressing the ATTN or BREAK keys. At this point you can enter any one of a series of commands. Some of these commands can be used to immediately change the format of your output beginning with the next output line. Below is a list of some of the commands you can enter. These commands are further described in *Chapter 5. MUSIC Commands*.

/CANCEL (or /CAN)
: This command immediately stops your program and its output and will cause you to return to command (*Go) mode.

/SKIP n
: This command is used to skip past output lines that you do not want to see displayed at your workstation. The *n* in this command is the number of lines you wish to skip.

/COMP
: The COMPRESS command will cause all sequences of multiple blanks characters to display as 1 blank.

/WINDOW
: This command can be used to cause only certain sections of each line to be displayed. For example, you could cause MUSIC to display only the first 20 characters of output on each line.

## Tabs

MUSIC supports both input and output tabs. The use of output tabs can greatly increase the effective printing speed of your terminal. Output tabs can only be used if the terminal is capable of supporting physical tab settings. These must be set to the same values as specified for the output tabs in the user profile or via the /TABOUT command. The IBM 2741, 3767 and 1050 terminals all have input and output tab capability. The IBM 3270 display terminal does not, though its output speed is so fast that it doesn't need them. Some TTY type of terminals support output tab settings. Input tabs can be useful regardless of the physical tabbing capabilities of the terminal. When inputting on terminals with tab settings, the TAB key is used to skip to the next tab column. In this case the data displayed at your terminal appears as it will when processed by the system. On terminals without physical tabs, a logical tab character can be chosen to provide the tab function, (usually a seldom used character). Pressing this character will not cause the terminal to skip to a specific

column, but when the data is processed, the system will align the data to the correct column. Refer to the writeup on your particular type of terminal for more information. The writeups on the MUSIC commands /TABIN, /TABOUT also contain more information about the use of tabs.

## Carriage Control

Each printed line on the terminal starts with a carriage control character. This carriage control character is not printed but is used instead to specify the number of lines to skip before printing the current one. Thus carriage control characters can be used to produce output that is single, double, triple spaced. The convention MUSIC uses follows the standard established for batch jobs. In this way, you can check out a batch job from a terminal and get the same spacing as it would on batch. Some extra carriage control characters have been added to MUSIC to provide more flexible support for some terminals. Refer to the description on carriage controls in *Chapter 4. File System and I/O Interface* for more information.

## Direct Terminal Control

MUSIC allows programs to intermix tab skips, line feeds, backspaces, etc., into normal output lines. Additionally, a special carriage control character is assigned so that you can output directly to the terminal bypassing the usual MUSIC translation and output control. (The subroutines NOTRIN and TRIN can be used to control this translation for terminal input.)

These facilities can be useful in some special applications particularly when it is required to control special terminal features such as dynamically setting output tabs, or triggering special graphical features on the terminal. These functions may require different handling depending on the type of terminal being used. Therefore, your program should first verify the terminal type by a call to the MUSIC system subroutine TSUSER to ensure proper output operation of your program.

# Workstation Error Messages

*Enter BREAK-time command (or blank line to continue)

The system was expecting one of the break-time commands and the user has entered something else. Typically, this BREAK mode was entered by the user pressing the BREAK or ATTN key. On ASCII (TTY) terminals, it can also be inadvertently entered by the user typing information into the terminal before the system is ready to read it.

*Invalid command

The command just entered is undefined or invalid in the current mode.

*Command not valid at this time

The command just entered is a valid command but cannot be used in the current mode.

*Invalid parameter

A parameter entered on a command is not defined for that command or has a value that is not in the correct range.

*Missing parameter

The command just entered requires a parameter that has not been included.

*Line too long, retransmit

A line longer than 80 characters has been entered, or an undefined tab setting has been used for input, or the total number of characters typed for the line (including backspaces) exceeds the limit of 100. For conversational reads during program execution, the limit is 250 characters including back-spaces.

*Invalid character, retransmit

An invalid character has been entered. This can be caused by a terminal transmitting incorrect parity. ASCII (TTY) terminals should be set up for EVEN parity transmission. Other parity bit combinations may also be acceptable.

*Rejected

The user has attempted to cancel a non-cancellable program.

*TRANSMISSION ERROR

An error has occurred on the telecommunications line, for either input or output. If received just after a line has been sent then the line will probably have to be retyped.

*No more sessions chained, Use /OFF

This Multi-Session error message occurs when you try to delete (F5 by default) the last session. There are no other sessions on your terminal at this time.

*Delete is only valid from *Go mode.

This Multi-Session error message occurs when you are trying to delete a session (F5 by default) but you are not in command (*Go) mode.

*Multi-session limit exceeded.

This Multi-Session error message occurs when you attempt to add more sessions than allowed in your user profile. This can also occur when the /DISCON command is used.

*Insufficient storage to create new session.

The system does not currently have enough storage to create the control blocks required for a new session.

*Function key not defined, application continued.

PA2 was pressed in full screen mode entering Multi-Session mode. The function key you pressed was not defined to move to a new session, so the request has been ignored and the original program resumed from where PA2 was pressed.

*There are no sessions chained.

This Multi-Session error message occurs when you have pressed either F7 (previous session) or PF8 (next session) when no other sessions are active.

# Chapter 3. Using Batch

# Batch Concepts

MUSIC processes jobs from a batch facility at the same time as it is processing jobs from workstations. This batch facility is usually located at the main computer site.

You can submit batch jobs directly from your workstation using the SUBMIT facility described below. Your site may have an alternate method or procedure for submitting MUSIC batch jobs.

Some installations allow MUSIC users to submit batch jobs to other operating systems they may be running and to retrieve output from the job. This is discussed later on in this chapter.

Since many other people may be sharing the same batch facility with you, you might have to wait for some time until your job is run. On the other hand, your MUSIC batch job can be working without you being present to supervise its progress. An installation may wish to give lower priority to jobs submitted to batch, so as to best serve the users who are running jobs from the workstation. Submitting a batch job to run over-night results in better service for all users during the day.

The batch facility features a high-speed line printer that can typically print 132 characters per line at speeds in the range of 1000 lines per minute.

MUSIC batch jobs have access to your files and User Data Set (UDS) files. In addition, batch jobs can use files located on reels of magnetic tape and mountable disk packs.

Since you are not generally present when your batch job is run, it is important to tell MUSIC about any special requests your job might have such as special paper that is to be mounted. You are required to provide maximum limits of how long your job is to run and how much printed output you expect. This information is useful to let the operators know what to expect from your job. This information is also a safeguard for you in case your job goes astray and wants to print thousands of pages of output instead of the ten pages you thought it would.

## MUSIC Commands and Statements on Batch

A number of MUSIC commands have meaning only when used from a workstation. Such commands include those to set input and output tabs and to skip output. MUSIC Job Control statements can be stored in a file. You may use all of MUSIC's job control statements from batch.

### Job Control Statements

The following is a list of all the job control statements that you can use from batch. Most of these statements are explained in this chapter. For additional information refer to *Chapter 6. MUSIC Job Control Statements*.

```
/COM
/DATA
/END
/ETC
/FILE
/ID
/INFO
/INCLUDE
/JOB
/LOAD
/OPT
/PARM
/PASSWORD
/PAUSE
/SYS
```

## MUSIC Commands

The MUSIC commands, which must be prefixed with a slash (/) for use with batch jobs, are listed below. Examples of their usage is explained in this chapter. For details about each command refer to *Chapter 5. MUSIC Commands*.

/LIBRARY
/PURGE
/SAVE

# Preparing a Batch Job

All MUSIC batch jobs have a /ID as their first statement and a /END as their last statement. A statement containing your special batch password can be located anywhere in between.

Any requests for special paper or other messages to the operator go on a /PAUSE statement which you can put immediately after the /ID statement

A typical batch job set up is shown below:

```
/ID ...      (userid and job limits)
/PASSWORD=ABC
/INCLUDE XYZ
/END
```

This example shows how to run the program XYZ that is stored in a file. The user's batch password in this example is ABC. MUSIC allows part of your program and/or data to be in a file or User Data Set (UDS) file.

# Format of the Batch /ID Statement

The first statement of your batch job must be a /ID. It must have information in specific columns as shown in Figure 3.1 below. Your job is automatically canceled if it exceeds any limit given on the /ID statement.

If your userid is not authorized to use the time specified on the /ID, then MUSIC automatically lowers it to

the maximum that you are allowed. Any time limit given on a /SYS statement is ignored when your job is run from batch.

```
1   5               21     >29          (column number)
|   |               |      |
/ID jobname         userid,tim,pgs,cds
```

*Figure 3.1 - /ID Statement for Batch*

| Columns | Contents |
|---|---|
| 1-3 | Always the characters /ID. |
| 5-12 | An optional name that you wish to give your job. This name has no effect on your program. |
| 21 | The userid followed by a blank or comma. |
| >29 | The Time, Pages, and Cards parameters follow the userid and start in column 30 or later. Allow 3 digits for each parameter and separate each with a blank or comma.<br><br>*tim* is the maximum job time in units of 60 service units. Thus 002 would mean 120 service units. The leading zeros are required -- blanks are not the same. If column 32 contains the letter "S" then the time limit will be in service units. Thus 20S would mean 20 service units. The word MAX can also be used.<br><br>*pgs* is the maximum number of pages of output. A page is typically 60 lines in length. A number of 999 in these columns means unlimited number of pages. Use leading zeros -- for example, type "050" for 50 pages not " 50".<br><br>*cds* is the maximum number of punched cards. A number of 999 in these columns means unlimited number of punched cards. Use leading zeros -- for example, type "050" for 50 cards not " 50". |

# Printer Control

The printed output of your job is controlled primarily by the carriage control characters in the output. These carriage controls can be used to cause the printer to skip to a new page, double space, etc. These carriage controls are discussed in detail in *Chapter 4. File System and I/O Interface*.

MUSIC automatically skips over the first few lines at the top and bottom of a page. This automatic skipping can be avoided for the duration of your job by specifying the option NOSKIP on a /SYS control statement.

# Special Operator Message - /PAUSE

A statement of the form /PAUSE text may be placed after the /ID statement to display the message *text* just before the job is run. The operator has to respond to this message before your job will continue. Only one /PAUSE statement is allowed per job. The text of the message can include tape volume names required by your job and any other special requirement that the operator must tend to.

## Return Location Messages

You may place one or more /ETC control statements immediately following the /ID or /PAUSE statement. These control statements will be printed on your output immediately following the /ID line. They can be used for special handling messages to the operator, after the job has been processed and printed, such as where your job is to be returned etc. Notice that these statements do not affect the processing of your job and that they are only printed after your job has been run. Requirements that are to affect the job must be specified on the /PAUSE statement.

## Purging Files

Files can be purged from batch. However, your installation most likely does not allow this operation. If purging is allowed, then the set-up is as follows:

```
/ID ...              (userid and job limits)
/PASSWORD=abc
/PURGE name
/END
```

*abc* is the user's batch password, and *name* is the name of the file to be purged. See the PURGE command for more details.

## Unit 9 on Batch

If your batch program tries to read from the workstation during execution, MUSIC will automatically interpret the request as a read from the reader instead. This means, for example, that FORTRAN programs that do a READ from unit 9 will be handled exactly as if it were a READ from unit 5.

# Submitting Jobs to MUSIC Batch & Other Operating Systems

MUSIC offers the ability to submit files for processing on MUSIC batch, or other batch processors accessible via VM. The facility also exists for sending the printed output from these jobs back to MUSIC for subsequent inspection. The various programs and commands involved in the submission of jobs and the processing of their output are documented together in this chapter of the manual. Since each MUSIC installation can setup the interfaces in different ways, you should consult your installation for details of what is available to you.

## SUBMIT Program

The SUBMIT program submits one or more files as a job to MUSIC batch, or to other available batch processors.

By default, SUBMIT generates the appropriate job control statements and submits file(s) to MUSIC batch. Parameters for these job control statements such as CODE, TIME, and PAGES, etc. can be overridden by parameters specified on the SUBMIT command or on a /INFO statement. Jobs can be submitted to other processors by specifying the processor name on a /INFO statement or in the TO parameter on the SUBMIT command. For example, the processor parameter of /INFO is similar to the TO() parameter of the SUBMIT command. Whatever can be specified on the /INFO statement can also be specified as parameters for SUBMIT.

### Usage of Submit

When you are in the Command (*Go) mode or in the Editor, you can enter the following to invoke the SUBMIT program:

```
SUBMIT fn1 [fn2] [fn3] . . . [kw1(value1)] [kw2(value2)] . . .
```

The shortest abbreviation for SUBMIT is SUB. The parameters within square brackets are optional.

### Parameters

fn1 fn2 fn3 etc.

These specify the names of the files to be submitted. At least one file name has to be specified. Under the Editor, the special name of **CUR* can be used to indicate the current contents of the file being edited. The first file *fn1* should usually contain a /INFO statement (see description later on in this chapter) specifying where the job is to be submitted. If more than one file name is specified, the files are concatenated (joined) in the specified order to form a single job to be submitted.

kw1(value1) kw2(value2) etc.

These are keyword parameters whose values are substituted into the control statements that are submitted with the job. These parameters are dependent on the particular processor to which the job is being submitted, and are usually used to specify items such as time and page limits, passwords, output destinations, etc. The special keyword parameter *TO*(*processor*)

can be used to specify the processor (system) to which the job is to be submitted. Consult your installation for a list of valid batch processors. If any keyword parameters are specified, they must follow the file name specifications above.

## /INFO Job Control Statement

The /INFO statement is used to inform SUBMIT where the job is to be submitted. Keyword parameters, as described under SUBMIT, may also be included on this command. The benefit of the /INFO statement is that the submit destination and the values to be used in the job control statements do not have to be specified each time the SUBMIT program is invoked. If /INFO is used, it must be specified on the first line of the job (i.e., the first line of the first file to be submitted). The syntax of the /INFO statement is the following:

```
/INFO processor [kw1(value1)] [kw2(value2)] . . .
```

The first parameter is the name of the processor (system) to which the job is to be submitted. It could be specified as *MUSIC* to submit the job to MUSIC batch. The keyword parameters *kw1, kw2,* etc. are identical to those described for the SUBMIT program. If the parameters do not fit on one /INFO statement, they can be continued on subsequent /INFO statements.

*Notes:*

1. If there is no /INFO present in the job and the *TO* parameter is not specified when the SUBMIT program is invoked, the job will be sent to MUSIC batch.

2. Any parameters specified on the SUBMIT command override the corresponding ones specified on the /INFO statement.

3. The first /INFO statement must include a processor name.

4. If the same keyword parameter is specified more than once on the /INFO statement(s), the last one will take effect.

## Submitting to MUSIC batch

If the processor name of *MUSIC* is specified, or if no name is specified, the job will be submitted to MUSIC batch. The following is a list of keyword parameters that can be used when submitting to MUSIC batch. The minimum abbreviation is given in upper case letters.

| Keyword | Description | Default |
|---|---|---|
| Userid | 1-16 character userid | Sign-on userid |
| Code | 4 character userid * | Sign-on userid |
| Sub | 3 character subcode * | Sign-on subcode |
| TIme | Time limit in units of 60 service units | 10 |
| PAges | Page limit | 500 |
| CArds | Punch card limit | 0 |
| CLass | 2 character job class (see description below) | AA |

| | | |
|---|---|---|
| FORms | Special forms code | Standard forms |
| COPies | No. of copies of printed output | 1 |
| Route | Route code for printed output | System Printer or the default specified in user's Profile |
| PW | MUSIC batch password | Must be specified if other than the default code is used |
| MSG | Message to operator | Must be specified for jobs requiring special handling or tapes |

* The "Code" and "Sub" keywords are available for compatibility with earlier versions of MUSIC. User-ids were restricted to 7 characters (4-char code with a 3-char subcode).

The CLASS parameter is used to specify whether the job requires special handling by the operator and when it should be executed. The following is a list of valid class codes:

AA No special handling. Run as soon as possible. This is the default.

SA Requires special handling. Run as soon as possible.

TA Requires tape mount. Run as soon as possible.

AO No special handling. Run after 6 p.m.

SO Requires special handling. Run after 6 p.m.

TO Requires tape mount. Run after 6 p.m.

The ROUTE parameter is used to specify the location for printed output when submitting jobs to MUSIC batch. The following names are valid:

SYSTEM Send the output to the VM system printer for processing.

MUSIC Send the output to the MUSIC OUTPUT facility (to be retrieved at your work-station).

rscsname *rscsname* is the name of a RSCS printer where the output is sent for processing. For example: R(PRINTER3) means the output is sent to RSCS and queued for printing on linkid PRINTER3.

The name of a MUSIC-controlled ASCII or 3270 printer can also be specified on the SUBMIT command.

# Examples of SUBMIT

The first example is shown in full. Only the actual SUBMIT command is given in the rest of the examples. It is assumed that there is no /INFO statement specified in the job for the first four examples.

1. Submit the file PROG1 to MUSIC batch.

```
*Go
submit prog1
103 RECORDS SUBMITTED.
```

```
*End
*Go
```

2. Submit the file PROG1 to MUSIC batch, overriding the default page and time limits.

```
submit prog1 time(10) pages(200)
```

3. The files X.PROG and X.DATA are submitted to MUSIC batch and the printed output is routed back to MUSIC (i.e., the workstation).

```
sub x.prog x.data route(music)
```

4. The tape job in the file ARCHIVE is submitted to MUSIC batch with a message asking the operator to mount the appropriate tape.

```
sub archive cl(ta) msg(mount archive tape 1)
```

5. If the first line in the file PROG is...

```
/INFO MUSIC PAGES(600) TIME(80) ROUTE(MUSIC)
```

then entering...

```
sub prog
```

in *Go mode would submit the file PROG to MUSIC batch to be executed as soon as possible. The limits for the job would be those set from the /INFO statement and the printed output is routed back to MUSIC.

6. If your file does not have a /INFO statement, then to achieve the same results as example 5, enter the following:

```
sub prog TO(MUSIC) PA(600) TI(80) R(MUSIC)
  or
sub prog PA(600) TI(80) R(MUSIC)
```

# OUTPUT Management Facility

The OUTPUT Management Facility allows you to inspect batch output from a workstation. Output from MUSIC batch jobs is sent to the OUTPUT facility by specifying the ROUTE(MUSIC) parameter of the SUBMIT command. Other operating systems such as VSE, MVS, and CMS can be set up to send printed output to MUSIC in this way. (Consult your installation for details.)

The OUTPUT Facility allows you to look at and work with files of print data in the MUSIC Print Queue. Each entry in the Print Queue represents the output of a previously submitted batch job, or data resulting from a PRINT command, or other print data sent to the Print Queue. Each Print Queue entry has a unique id number (Idnum), an owner, a name (for example, the job name of the batch job), a route (destination) name, and other attributes.

A systems administrator can view all items in the queue (not just their own jobs). Refer to the *MUSIC/SP Administrator's Reference* for more details.

When you start the OUTPUT Facility (by typing the command "output"), the Print Queue entries you own are displayed on the screen. If there are more entries than can fit on the screen, you can use function keys F8 (next page), F7 (previous page), F4 (top), and F5 (bottom) to move around. You can change some of the attributes of an entry by typing over the fields on the screen. You can work with an entry by typing a request code in the "Req" area at the left of the entry; request codes are V (view the data file), B (browse the data file), C (copy the data to another file), P (print), D (delete), etc. The available request codes and PF keys are summarized at the bottom of the screen. You can also enter an OUTPUT or MUSIC command in the command area provided on the screen. You press F3 to leave the OUTPUT Facility.

You own a Print Queue entry if its owner field is the same as your file ownership id (your userid, without the subcode if any).

Entries with route name MUSIC or HOLD are not scheduled to print. They will be held in the queue until you change the route name or delete the entry. Entries with other route names are usually waiting to print, and will soon be printed and removed from the queue. Any entry that has been in the queue for n days or more is automatically deleted; n is set by your installation (typically n is 10 or 20).

## Using the OUTPUT Facility

When your workstation is in *Go mode, the OUTPUT facility is invoked by entering "output". The following screen appears.

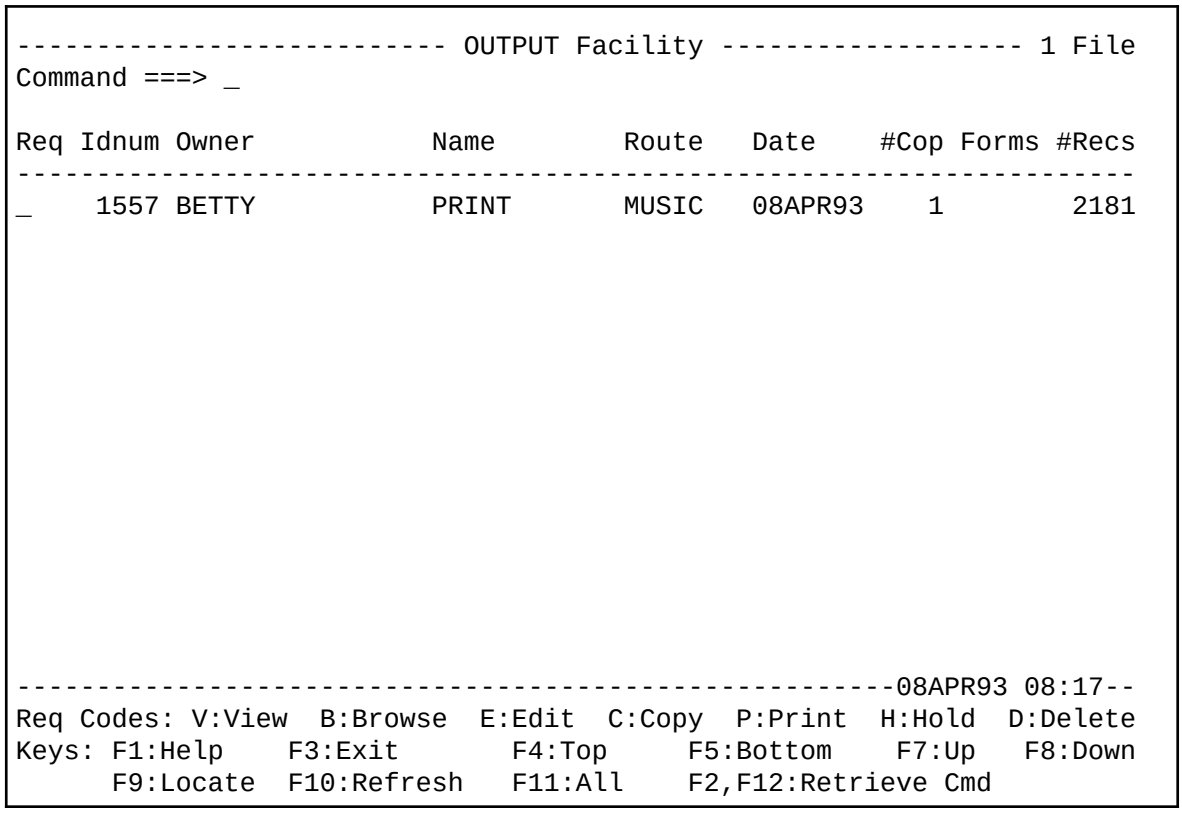



*Figure 3.2 - OUTPUT Facility Screen*

## OUTPUT Screen Description

The figure above illustrates the screen display for the OUTPUT facility. Each field on the screen is described below. (If your workstation does not support full-screen applications, see the topic "Non-3270 Support (TTY Mode)" later.

Command => A command can be entered in this field. There are two types of commands: OUTPUT commands and MUSIC commands. An OUTPUT command is a special command recognized by the OUTPUT Facility, such as LOCATE, SORT, TOP, BOTTOM. Any other command is assumed to be a MUSIC command and is executed as such. You can indicate a MUSIC command by typing a slash (/) before the command name. For example, "/edit myfile". The / is required only if the command name is the same as the name of an OUTPUT command.

Req Enter a request code in this field, beside the Print Queue entry you want to work with. Request codes are one letter abbreviations for certain operations, such as V (view the data file), B (browse the data file), P (print), D (delete), etc. Available codes are listed at the bottom of the OUTPUT screen. You can enter several request codes, for several entries. For example, to view 3 entries and delete another, enter 3 V's and a D. When you press Enter or a PF key, the requests are done in order, from top to bottom of the screen.

Idnum This is a unique, fixed identification number assigned to each Print Queue entry. You cannot change an entry's Idnum.

Owner This is the 1 to 16 character file ownership id of the MUSIC user who owns this entry. The ownership id is the MUSIC userid, but without the "subcode" part (if any). Only a system administrator can change an entry's owner field. For a print file spooled to MUSIC from

some other system, the owner field is the VM distribution (DIST) field.

Name — This is a 1 to 12 character name associated with the Print Queue entry. For the output of a batch job, it is the job name. For an entry generated by the PRINT command, it is PRINT. For an entry generated by a /FILE statement in a job or program, "/FILE ddname PRT(route)", it is the ddname. For a print file spooled to MUSIC from some other system, it is the "CMS filename" field. You can change the name by typing over it on the screen. The change takes effect when you press the next action key (Enter or a PF key other than F3).

Route — Indicates on which printer the file is to print. You can modify this field to have the job print at another printer, provided it has not yet started to print. The route name is 1 to 8 characters. It must be a valid route name defined for your installation. The special names MUSIC and HOLD normally indicate entries that are not scheduled to print at this time. For other names, the entry and print data are automatically deleted once the job is printed on the indicated printer.

Date — The date (ddmonyy) the print file was created. NOTE: After an entry has been in the Print Queue for more than n days, it is automatically deleted (along with its data file). The number n is defined by your installation. Typical values for n are 10 or 20.

#Cop — Specifies the number of copies to be printed (1 or more). You can modify this field by typing over it.

Forms — Specifies the forms number or name (1 to 8 characters), if the output is to print on special forms. A blank in this field indicates standard forms. You can modify this field. (See your installation for names of special forms.)

#Recs — The number of records (lines of output) in the data file.

## OUTPUT Request Codes

A list of available request codes is found at the bottom of the OUTPUT screen. These codes can be entered beside jobs in the "Req" field on the screen.

B (Browse) — Enter a "B" beside the print files you wish to browse. This uses the editor (in BROWSE mode) to display the print data file. See also V (view).

C (Copy) — Enter a "C" beside the print file(s) you wish to copy to your own file(s). You will be prompted to enter the name of the file to which the print data is to be copied. If the target file already exists, you are asked whether to replace it; answer Y (yes) to replace the old file, or N (no) to cancel the copy, or A (append) to copy the print file to the end of the existing file. After a copy operation, you can append other print files to the end of the same target file by entering " instead of a file name, when prompted by COPY; this is useful when you want to combine two or more print files into a single file.

D (Delete) — Enter a "D" beside the print files you wish to delete. Both the Print Queue entry and the corresponding data file are deleted. Be careful when using this request code, since a print file CANNOT be retrieved after it is deleted!

E (Edit) — Enter "E" beside the print files you wish to edit. Note that you cannot use the editor to make changes to the print file, since it is not in your library.

H (Hold) — Enter "H" to put print files on hold. The route location changes to HOLD.

P (Print) — Enter "P" to schedule a file for printing. You can change the route name, if you want, by

typing over it on the screen; the P request code uses the new route name if you do so. The P request is handled according to these cases:

1. If the route name is MUSIC or HOLD (or equivalent), the route name is changed to your default route name. If your default route name is a printer name, the file will be scheduled for printing on that printer, and the file will be deleted after it is printed.
2. If the route name is a name such as PC1, indicating printing on your local PC printer or PC network printer, the OUTPUT Facility tries to print the file immediately. The file is not deleted after printing. You should use the D request code to delete it yourself when you no longer need it.
3. Some other route name normally indicates an actual printer. For these, no action is taken, since the file is already scheduled for printing.

V (View) Enter "V" to view print files. This uses the VIEW program to display the print data file. See also B (browse).

## OUTPUT Function Keys

The "Keys" section at the bottom of the OUTPUT screen summarizes how the function keys are used in OUTPUT.

F1 - Help Provides help on how to use the OUTPUT facility.
F2 - Retrieve Same as F12 (see below). Retrieves a previous command.
F3 - Exit Exits from the OUTPUT Facility. PA1 has the same effect.
F4 - Top Moves to the top (beginning) of the list of Print Queue entries.
F5 - Bottom Moves to the bottom (end) of the list of Print Queue entries.
F7 - Up Displays the previous screen of Print Queue entries.
F8 - Down Displays the next screen of Print Queue entries.
F9 - Locate Locates the next occurrence of the string specified by the previous LOCATE command. For example, if you previously entered the command "locate fred" in the command area, pressing F9 locates the next occurrence of "fred".
F10 - Refresh Refreshes the screen with the latest contents of the Print Queue. When OUTPUT starts, it reads the Print Queue and remembers the entries it found. Those entries are displayed, until you press F10 to tell OUTPUT to read the queue again. Refresh may show new entries recently added to the queue, or drop entries that have recently been deleted.
F11 - All (This key is ignored unless you are a system administrator with the LSCAN privilege.) It displays all entries in the Print Queue, for all users. It is equivalent to the command SCAN FLIP. To revert to the normal display, press F11 again.
F12 - Retrieve Displays the previous command entered in the command area each time this key is pressed. Up to 10 previous commands can be recalled. When you press F12, any old text in the command area is replaced, without being executed. After retrieving a command, you can modify if you want, then press Enter to execute it, or Clear to clear it from the command area, or F12 to retrieve the previous command.

# OUTPUT Commands

The following special OUTPUT commands can be entered in the command area of the OUTPUT screen. If a command you enter is not recognized as one of these, it is executed as a MUSIC command. You can type a slash (/) before a command to indicate that it is to be treated as a MUSIC command, not an OUTPUT command. You can use the Clear key or F12 to move the cursor to the command area.

The minimum abbreviation is shown below each command name.

BOTTOM

B — Displays the last full page (screen) of Print Queue entries.

LOCATE string
L — Locates the next Print Queue entry that contains the specified character string. The string may appear within any of the displayed fields: Idnum, owner, name, route, date, etc. For example, the command "locate fred" would find an entry with owner FREDDY, or with name JOBFRED, etc. If you do not specify a string, the string from the previous LOCATE is used (this is equivalent to pressing F9), and the next occurrence is located.

SORT field order
SO — Sorts the displayed entries in ascending (A) or descending (D) alphabetical order by a specified field.

The first parameter, field, is one of the following (the first 3 characters of the field name may be used as an abbreviation):

| | |
|---|---|
| IDNum | Sort by Idnum |
| OWNer | Sort by owner |
| NAMe | Sort by name |
| ROUte | Sort by route name |
| DATe | Sort by date (chronological order) |
| COPies | Sort by number of copies (numerical) |
| #COpies | Sort by number of copies (numerical) |
| FORms | Sort by forms name |
| RECords | Sort by number of records (numerical) |
| #REcords | Sort by number of records (numerical) |
| OFF | No sorting (equivalent to sort by Idnum) |

The second parameter, order, is A or D. If omitted, A is assumed.

If you want to sort by a field (say ROUTE), and sort by a second field (say NAME) within that, use two consecutive SORT commands:

```
sort route
sort name
```

TOP
T — Displays the first page (screen) of the Print Queue entries.

## Order of Operations on the Screen

It is possible to combine several operations at the terminal before pressing an action key. Operations are done by the program in the following order:

1. Print queue entry changes (route name, number of copies, etc.) and request code action, from top to bottom of screen. For a given entry, request code action is done after the entry changes.

2. Command from command area, if any.

3. PF key, if any.

When Clear, F3 (Exit), or PA1 is pressed, any changes to the screen since the last action key are ignored.

# Non-3270 Support (TTY Mode)

When the OUTPUT Facility is used on a terminal that does not support 3270-type full-screen applications, a different user interface, called TTY mode (or non-3270 mode), is used. It is a line-at-a-time interface, with commands used for all input from the user. Additional OUTPUT commands are available, to provide the functions done by screen input and PF keys in 3270 mode.

For more information enter the MUSIC command: "HELP OUTPUTTTY".

# Printing Files

Files can be printed on any of the line printers defined in the system by using the PRINT command. The PRINT program can be invoked in *Go mode or when you are in the Editor.

The PRINT command schedules the printing of a file to a specified printer. The file name must be the first parameter. The other parameters are optional and may appear in any order. Carriage control is added to skip to a new page every sixty lines unless CC is specified or the file's record length is 121 or 133.

The PRINT command does not send your file for execution. It prints the contents of the file. The Editor also has an similar PRINT command.

**Syntax**

```
PRINT filename [ROUTE(printername)] [FORMS(x)] [COPIES(n)] [CC  ]
               R                    F          C          [NOCC]

               [PAGELEN(m)]
                P
```

If you are in the Editor, the shortest abbreviation for PRINT is PRI. There is no abbreviation in *Go mode. The parameters within square brackets are optional and may appear in any order, separated by a blank.

## Parameters

filename — The name of the file to be printed. Under the Editor, the special name **CUR* indicates the current contents of the file being edited, and the special name . (period) indicates marked lines.

printername — The name of the printer where the file is to be printed. This may be an actual printer name or a printer location. It is 1 to 8 characters long. Some documentation refers to this as a "route name" or "routing name". The names are assigned by your system administrator.

If you do not specify a printer name, a default name is used. If you have used the command "ROUTE printername" previously in this MUSIC session, that name is the default. Otherwise, the name defined by ROUTE(name) in your User Profile (the PROFILE command) is used, if any. Otherwise, a default name based on your workstation location may be used. If none of the above cases apply, the name SYSTEM is used.

The following names are valid:

SYSTEM — Sends the output to the standard system printer.

MUSIC — Sends the output to the MUSIC Output Queue (the OUTPUT Facility).

DUMMY — Discards the output. Nothing is printed.

rscsname — The name of an RSCS printer. For example: R(PRINTER3) means the

output is sent to RSCS and queued for printing on linkid PRINTER3.

prtname — The name of a MUSIC-controlled ASCII or 3270 printer as defined by your installation. Consult your installation for a list of valid names.

PC1 — A printer name such as PC1 may be defined by your installation. It prints the file on your PC printer (using DOS device LPT1), provided your PC is connected to MUSIC via NET3270 or PCWS. If the connection does not support PC printing, the data is sent to the MUSIC Output Queue (the OUTPUT Facility). Similarly, PC2 uses device LPT2 and PC3 uses device LPT3. When printing to a PC printer, some PRINT parameters such as COPIES, FORMS and PAGELEN may be ignored.

n — The number of copies that should be printed. The default is 1 copy.

x — A one to 8 character string indicating which forms are to be used when printing the file.

m — The number of lines per page. The default is 60 lines. This parameter is used only when the NOCC parameter is in effect.

CC — If specified, it indicates that the file to be printed already contains a carriage control character on the first character of each line in the file. The PRINT command attempts to honour these control characters.

NOCC — Indicates that the file does not contain printer control characters. The file is printed single spaced, with a skip to a new page after each *m* lines of output. NOCC is the default, except when the record length of the file is 121 or 133, in which case CC is assumed.

**Example:**

```
*Go
print file1
*In Progress
15 records scheduled to print, route SYSTEM
*End
*Go
```

# Checking the Status of Printers

The PQ utility program can be used to find out what is queued to print on the various printers. It can be invoked from *Go mode by entering PQ. It should be noted that if the printer in question is under the control of another operating system, the fact that MUSIC has transmitted the print file to that system and removed it from the print queue does not necessarily mean that the print file has actually been printed. The OUTPUT command is useful in displaying the status of the individual print files.

# Chapter 4. File System and I/O Interface

# File Systems

MUSIC supports two file systems -- Save Library files and the User Data Set (UDS) files. Most users use only the Save Library file system. Unless otherwise specified, reference to files on MUSIC/SP refer to Save Library files.

Files are easy to create, edit, and manage. They can contain programs, text and data of any kind and sequential, direct and indexed access is supported. Programs can dynamically open and close files. The contents of files can be easily exchanged with other host systems and personal computers.

UDS files use the IBM standard VTOC format established around 1965. It does not offer the same performance, data compression and usability benefits found with the regular MUSIC files. It is mainly used in MUSIC to handle some temporary scratch space and to allow concurrent data sharing with other operating systems in some circumstances. Most MUSIC users will never have to know anything about UDS files. UDS files are created and accessed through the /FILE statement. See *Chapter 5. MUSIC Commands* for more details about the /FILE command that applies to UDS files.

## Save Library Files

The Save Library file system (disk) is the main file system on MUSIC. Each file in the Save Library is owned by a particular user. File ownership is based on the ownership id which is the sign-on userid excluding any subcode. If two users share the same userid (different subcodes) then they also share the same files.

You normally have unrestricted access to the files you own. You can share files with other users if the appropriate access controls are specified.

Files can be created using the editor, using /FILE statements, or dynamically by application programs. A file name is assigned to the file when it is created and used to reference the file later on. To edit a file, the file name is specified as a parameter on the edit command. To run a program stored in a file, the file name itself is entered as a command. The LIBRARY and FLIB commands can list the names of all the files that you own. It will also optionally give details as to the size of your files, when they were created, and various other information. See *Chapter 5. MUSIC Commands* for more details about the LIBRARY and FLIB commands and other commands that apply to files.

### File Storage Technique

The Save Library is located on disk. Though physically consisting of multiple data sets, the user need never be concerned about where it is located. All users get their Save Library space from a common pool. When a file is purged, the space is automatically released and is made available for other users. The next user of the space you once occupied will be unable to look at what you had once stored there. Information in files is normally stored in compressed format in order to reduce disk space and disk read/write time. This compression is done automatically by the system and need not concern you.

### File Names

The user specifies a *file name* when a file is created. The file can then be accessed in future by merely giving the file name.

MUSIC allows the user great flexibility in the choice of names for files. Each file name can be up to 17 characters in length. File names cannot include blank characters. Each character of the name can be any letter (A-Z) or any number (0-9) or any of these special characters:

```
~  !  @  #  $  %  &  _  +  .
```

The exception to the above rule is that the first character of the name cannot be any of the following:

```
~ ! % & _ + .
```

and should not be the @ symbol.

*Notes:*

1. Files starting with @ are reserved for possible use by system programs and utilities. Some special rules apply to how they are handled so avoid their use. For example, the name @LIB is used by the LIBRARY command.

2. UDS file names cannot use the following special characters:

```
~  !  %  &  _  +
```

Examples of valid file names are: PROG, PROG.V1, RM.LETTER.05DEC92, REPORT_1, 51980, THISISALONGNAME.

Users who will be saving lots of files may wish to establish some naming convention to help identify the contents of each file. One suggested naming convention is to use names such as PROG.S to contain the source for the program PROG and PROG.OBJ to contain the object module for PROG. If a load module file is needed, it could be named PROG.LMOD. (The TAG command of the Editor can be used to store a descriptive phrase with each file to further help in identifying its contents.)

## Hierarchical Tree Structured File Naming

MUSIC supports the popular hierarchical tree structured file naming convention. It is used in the same way as the one used on personal computers running the DOS system. It is also compatible with the one used with the AIX and UNIX systems.

The tree structured method is particularly useful when you have many files. The following describes the benefits and the basic outline of how to use this naming convention. The use of this method is optional on MUSIC. If you do not use this method, then you are using what is called the "flat file system" naming convention.

Suppose you are a university professor. You teach several courses and you work on several research projects. You want to keep your files grouped by subject. You could establish a naming convention that your file names starting with the characters C100 referred to course C100, the ones starting with C200 to another course and VR would mean your files about your research project in virtual reality.

You could use names like C100.NOTES and C200.NOTES to keep the files separate between your two courses. Alternately, you could use the features of the MUSIC system to keep the files separate. You would do this by making "directories" for each course. One could be called C100 and the other C200. You make the directory for the C100 course by issuing the "MD C100" command. The other is made by using the "MD C200" command.

When you want to work on course C100, you change the directory by issuing the "CD C100" command. MUSIC will respond with the prompt of "*Go \C100>" indicating that you are in that directory. Now you

when you issue the "EDIT NOTES" command, the system knows you want the one associated with the C100 course. By default when you create new files, they will go into your "current directory" (C100 in this case.) You can issue the "DIR" command to list all the files stored under this directory.

Note that you can have a file called NOTES under different directories. They will all be different. In that way, your notes for course C100 are kept separate from the C200 course. The system does this by prefixing the name of your current directory in front of the files you use. So for example, editing the file NOTES under the C100 directory causes the system to work on the file "\C100\NOTES". You could edit the same file by typing in the command "EDIT \C100\NOTES" but that involves typing more characters.

Suppose you are editing the file NOTES as described above and you wanted to merge in the notes from the C200 course. This can be done by issuing the command "MERGE \C200\NOTES".

You can even define directories within directories. That allows you to further organize your files. Suppose you setup a directory called "VR" to contain the files to do with your virtual reality project. You want to have separate directories for your research results, the papers that you are writing about your research, and the letters asking for additional grants to support your project. Now you have a collection of directories which are linked in a hierarchical manner. You can use the MUSIC TREE command to help you visualize the organization of these directories. See the following diagram. This structure will resemble a tree with each directory being another branch in the tree. You travel around the tree by issuing the CD command to change the current directory. You can issue the command "CD \" to get back to the "root" of the tree where you have no current directory prefix being added to your files.

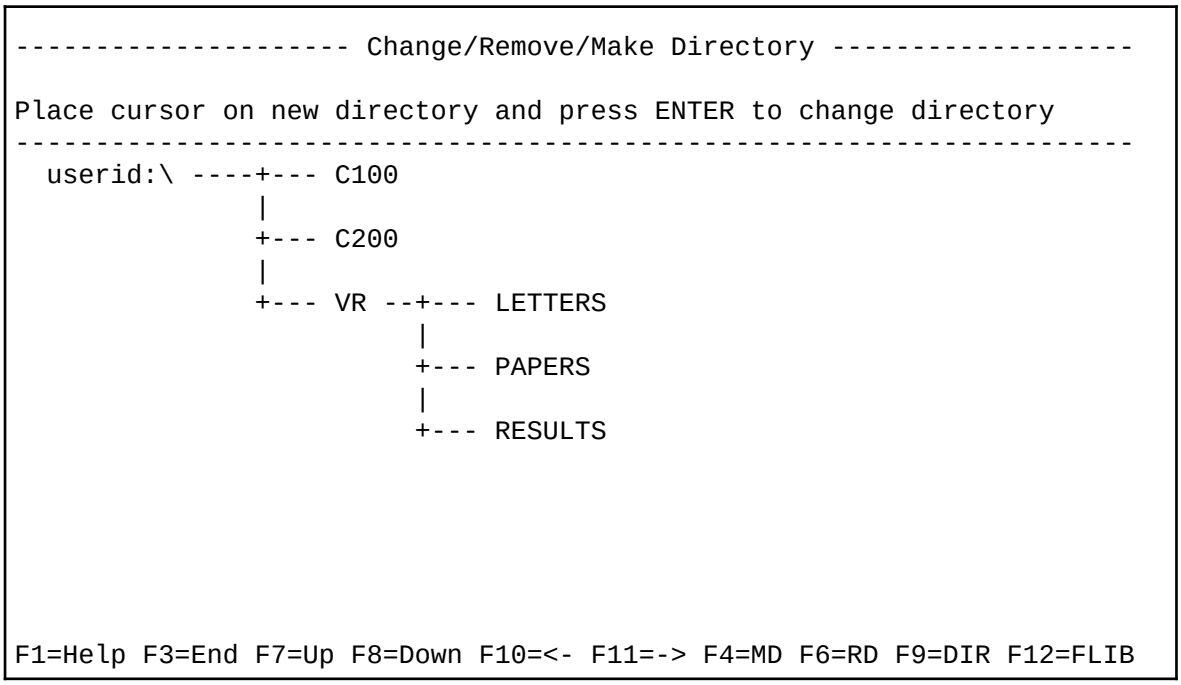



*Figure 4.1 - Screen display for TREE command*

Suppose you are in directory "\VR\LETTERS". You could have got there by issuing the "CD VR" followed by the "CD LETTERS". Now suppose you want to go work on your files in the C100 directory. You do that by issuing the "CD \C100" command. If you had issued the command "CD C100" the system would have thought you meant go to directory "\VR\LETTERS\C100" The "\" character at the beginning of the name is a signal to the system to go to the root before searching for the requested directory.

To remove any directory use the RD command.

There is a limit of 50 characters to hold the file name plus the directory prefix on MUSIC. That means that

you can have several levels of directories within directories but not an infinite number of them!

You can learn more about the CD, MD, DIR, RD and TREE commands by consulting the writeup on each command in *Chapter 5. MUSIC Commands*.

As mentioned under the previous topic, you should not use file names starting with "@". Those names bypass the use of the current directory prefix.

## Common and User Indexes

The system maintains an index (list) of all files that the user creates in the Save Library. A different index is maintained for each user. This index may be listed by using the LIBRARY command.

In addition the system maintains a common index to facilitate the sharing of files among different users. Some users are allowed to place a file name in this common index so long as no other user has previously used that name (see "Access Control" below).

*Note:* File names for MUSIC programs and commands are in the common index. If you have a "private" file with the same name as a "common index" file, then you will access your file only. Use the RENAME command to rename your file if necessary.

## Search Order (File Names)

When a user refers to a file, the system will first check the user's index to see if the file exists. If not, then the common index will be searched. This feature allows users to use other people's public files as simply as his or her own.

If the user has used the CD command to specify a current directory then that directory is searched first. If the file is not found there then the user's root directory is searched. If still not found then the common index is searched. (On MUSIC, this searching is a very fast operation. It does not read through lists of files like many other systems must do.)

If a file name has the share (SHR) attribute, it is possible to prefix a file name with an ownership id (userid excluding any subcode) as in the example OWNERID:PROG to directly refer to another user's file. Using a file name such as *COM:PROG will cause a search of the common index only.

The user's current directory will not be searched if the file name is prefixed with the character "\", or includes the owner's userid as in the example "OWNERID:PROG", or starts with the special character "@".

## Access Control (File Attributes)

Access to any file is controlled by control information stored with each file. This control information is set depending on the attributes you give when the file is created. The most commonly used ones are described below.

Saving a file with the *private* attribute means that only the owner of the file can access it. This is automatically the default. No entry in the common index is made in this case.

Saving a file with the *public* attribute means that other users can read the file. It may only be modified or deleted by its owner. This allows programs and data to be shared among many users while preventing unwarranted destruction of information. The name of the file is placed in the common index.

Saving a file with the *share* attribute means that other users can read the file but they must prefix the file

name with the owner's userid as in the example USERID:PROG. The file may only be modified by its owner. The name of the file is **not** placed in the common index.

The *execute-only* attribute can be used to further restrict access. It means that programs can be executed but not accessed in any other way. For example, a file with the public and execute-only attributes can be shared among many users while preventing inspection and copying it.

Some users may be restricted from saving files with public and share attributes. This is usually done to discourage file sharing between users and to avoid the creation of nuisance files in the common index.

## Record Formats

The record format is used to specify the format and characteristics of the records in the file. MUSIC supports the following types of record format for files:

F Fixed Format. All records of the file have the same record size.

FC Fixed Compressed. All records of the file appear to have the same record size. Internally, the file is compressed, with four or more consecutively repeated characters reduced to two bytes, to save disk space and disk read/write time. This type of record format is the most commonly used one.

V Variable Length. All records of the file may have different length.

VC Variable Compressed. All records of the file may have different length. Internally, the file is compressed, with four or more consecutively repeated characters reduced to two bytes, to save disk space and disk read/write time.

U Undefined Format. This is used by some system utility programs that do their own record handling.

## Record Size

The maximum record size of any file is 32760 bytes long.

## File Quotas

Your installation may have set a limit to the total amount of space your userid can use in the Save Library.

The biggest file you can create is 56 million bytes. Your installation may have set a lower limit than this for your userid.

The LIBRARY command can be used to show the amount of space used by each file and the total used by all your files.

*Note:* In addition to MUSIC commands there are several utility programs available to help you with files. See *Chapter 10. Utilities* for more information.

# Unit Numbers

The MUSIC system can read and write from many different places. For example, it can read from a disk file and write on a high-speed batch printer. Unit numbers are used in many cases to simplify the specification of what devices are to be used.

FORTRAN programs and utilities can be made to read and write from different units simply by using different unit numbers. For example, unit number 6 can be used to direct output to the workstation. The MUSIC default unit number convention is detailed below. The user may alter these specifications through the use of /FILE statements.

(COBOL, PL/I, VS ASSEMBLER and similar programs use *data definition names* (ddnames) instead of unit numbers. These processors have default ddnames. Refer to the write ups on these compilers for information about these defaults.)

6 Unit 6 is the printer. If the program is being run from a workstation, then specifying this number will cause the output to be directed back to your workstation. If the program is being run from batch, then this unit number directs the output to the batch printer. The maximum record size is 250 with the first character being a special carriage control character. Refer to the following discussion on carriage controls for further details.

7 Unit 7 is the card punch (it is rarely used anymore). This unit number is mainly used if your job is being run from batch. The maximum record size is 80.

5 Unit 5 is used to read files. The /INCLUDE statement can be used to tell MUSIC to read from the named file. Your job may use many /INCLUDE statements if you wish. (/INCLUDE statements may occur anywhere in your file.) The maximum record size is 80.

9 Unit 9 is used to read input conversationally from your workstation. Your job will temporarily pause waiting for you to respond to the read. From batch, a read on this unit number is taken as if it were one on unit 5. The maximum record size is 250. Users on 3270 terminals can only enter a maximum of 80 characters.

10 Unit 10 can be used to temporarily hold output from a program until it can be saved when the job is over. You can save the output from this unit with the MUSIC command "/SAVE name,SV" when your workstation is next in *Go status. The maximum record size is 80. See the discussion later on the Holding File for more details about this file.

# Carriage Control Characters

## Overview

The first character of each line printed via default unit number 6 is considered to be a carriage control character. This carriage control character is not printed but is used instead to specify the number of lines to skip before printing the current one. Thus carriage control characters can be used to produce output that is single, double, triple spaced, etc. MUSIC has extended the list of valid control characters to provide more flexible support for some workstations.

If an invalid carriage control is detected, the line will be effectively shifted over by 1 and a blank carriage control will be assumed. This automatic handling of incorrect carriage controls is performed merely as a convenience to the user. As other carriage controls may be defined from time to time, the user should not rely on this automatic correction procedure.

## Supported Carriage Controls

The following lists the supported carriage control characters and their effect. Some of them have particular meaning only on certain types of workstations. Some use a non-printable *hexadecimal* carriage control character to avoid possible accidental conflicts with incorrect carriage controls.

| | |
|---|---|
| blank | A blank for a carriage control results in normal spacing. |
| 0 zero | Double spacing. This control will print a blank line first before printing the current line. |
| - minus | Triple spacing. This control prints two blank lines before printing this one. |
| 1 one | Skip to new page. This control is supported on batch. Many workstations do not support an equivalent feature, in which case, this control will be taken as equivalent to a double space request. IBM 1050 terminals may be able to perform this operation. IBM 3270 terminals will cause the current line to be displayed at the top of the output area. The 3270 will go to MORE status first if the output area contains some output from previous writes. Certain TTY video terminals will work similarly to the IBM 3270 terminals. |
| + plus | Overprint. Supported only at batch. All terminal types will handle this as if it were a blank carriage control. |
| hex '70' | Erase Screen. Causes an immediate erase screen operation on 3270 terminals and certain TTY video terminals. On all other workstations, it will be taken as a blank carriage control. This carriage control may not clear the screen until some further output is generated by the program. Simply writing a line that is hexadecimal 70 followed by a hexadecimal 00 will be sufficient to clear the screen if no other program output is generated. This carriage control should not be used on batch. |
| hex '71' | Write to the message area (for 3270-type workstations only). The message area is the last line of the screen. The output line will be indented 9 spaces and the maximum length is 60 characters. An output line longer than 60 characters will be truncated. |
| hex '72' | Compress multiple blanks to a single blank, and also remove any blanks at the beginning of the line. The resulting line is displayed using single spacing. This control character is |

effective only for workstations, and should not be used on batch.

hex '79' Write to the input area (for 3270-type workstations only). The first 80 characters following the control character are written to the 3270 input line. This carriage control should not be used on batch.

hex '41' Direct output control. Causes the characters following this special carriage control to be transmitted to the workstation without the usual checking, translation and idle character generation. Since translation is not done, the output sequence must be in the required code for the specific type of workstation involved. This is used as part of the direct workstation control feature of MUSIC. Sending incorrect sequences may result in messages such as `*TRANSMISSION ERROR` or may even cause your line to be disconnected. This carriage control should not be used on batch.

hex '62' Sounds the alarm (beep) on a 3270-type workstation.

# Holding File

WARNING: The holding file is now of limited use and is provided mainly for compatibility for users accustomed to older versions of the MUSIC system. A /FILE can be used to specify that the unit 10 output is to be directly written to a file.

Users have access to a special holding file during the execution of their programs. This file can be used to hold the object modules produced from the compiler when the DECK option is used from a job running at a workstation. A user's program can also write output to this file via a write on unit 10.

The holding file is actually a file that is stored under your userid. It has the special name of @HOLD. (If your userid includes a subcode then the name is @HOLD.sub where sub is your 1 - 8 character subcode.)

After the job is finished, you can then save the contents of this file. You do this by using a command like "/SV name" or "/SAVE name,SV".

Alternately you can refer to it by the special name of /HOLD. This allows you to type in commands such as "RENAME /HOLD myobj".

A program can automatically cause an appropriate save command to be issued when the job ends. The system subroutine SAVREQ can be used for this purpose. Even if such an automatic save operation should fail, you still have the chance of entering a correct save command. (Use of the system routine NXTPGM does not give you this second chance.)

# Spooled Conversational Reads

## Introduction

MUSIC's normal conversation read facility (default I/O unit number 9), offers the user the maximum degree of interaction between the user and the running program. When a read 9 is performed, the program is stopped until the read is complete. At this time, the job resumes execution and can process the information just entered at the workstation. The program can decide whether to perform another read operation, or issue a message, etc.

Sometimes this degree of interaction is not required. Suppose for example, a program reads 10 lines in succession from the workstation and then processes this information. Using normal conversational reads will mean that the program will be interrupted 10 times -- once for each read. Only one interruption is needed. MUSIC addresses this requirement by providing the *Spooled Conversational Read* facility. This facility can be used to read a number of lines at a time while only requiring one interruption.

Considerable performance improvement can be achieved using this spooled facility since your job will require fewer time-slices. Additional flexibility is provided by the optional ability to process /INCLUDE control lines that may be entered at this time. A maximum number of lines to read in this mode can be given, or the user can end the set of reads by entering a blank line.

The MUSIC Editor is an example of a program which uses this facility. It uses spooled conversational reads when it is in INPUT mode for older models of terminals (non-full screen).

## Usage

The spooled conversational reads are always done via the MUSIC I/O unit number 5 not 9. The maximum record length is 80. This facility is not available when the job is run at batch. Also, use of spooled conversational reads cancels the effect of any previous call to subroutines NXTPGM, SAVREQ, and SIGNOF.

The program can detect the end of the set of lines read from the workstation by checking for the blank line or an end-of-file condition. The blank line will be present except when the reads were stopped by the program-specified line count, or it exceeded the maximum that could be stored in the system spool file. (This system spool file can hold at least 300 lines, so this should not pose much of a limit. You can, of course, re-issue the spooled read request to read in another set of 300 lines in this case.)

This spooled read facility is requested by the program via a call to the system subroutine SYSINR before each set of lines is read. This SYSINR subroutine is also used for other dynamic read facilities. The following gives the usual calling sequence. For further details consult *Chapter 9. System Subroutines* of this manual.

```
CALL SYSINR('/TRMIN  ',ln,lln,nc,k)
```

ln This number is normally given as 1.

lln This number is the maximum number of lines to be read. The number -1 specifies no fixed limit specification.

nc This number is normally 1.

k This number is usually given as 2 or 3. The value 3 means that /INCLUDE statements will be processed if found. The setting of 2 or 3 will give an end-of-file indication at the end of the set of lines read. Add 4 to get control of errors.

# User Data Sets (UDS)

User data sets are used to conveniently hold large files consisting of data, programs or load modules. A single file is limited only by the size of an entire disk volume. This type of file can be accessed by FORTRAN Direct Access.

UDS files can be allocated as temporary or permanent. Permanent files can be removed (scratched) when you are finished with them. Temporary files are used for the duration of one job only.

UDS files can be manipulated by using the Editor, a user's application program, or by utilities and subsystems provided with MUSIC.

Batch users can access UDS files stored on magnetic tape or those located on mountable disk packs.

You need a /FILE statement in any program that refers to a UDS file. Each record of a UDS file can be up to 512 characters in length. This *record size* is specified on your /FILE statement. Some processor or application programs may impose a lower limit to this record size.

## File Storage Technique

UDS files are stored on disk just like direct access data sets are stored by the OS and MVS operating systems. They have standard VTOC entries and data is not compressed on them by the system. The minimum file size is one track long.

## UDS File Names

Each UDS file is referred to by a character data set name (dsname) of up to 22 characters in length. In addition a disk pack volume name must be given. There are two types of dsnames:

Type 1 A dot (.) is used to separate the ownership id from the name. The dsname can be up to 22 characters long. For example: "GEORGEW.TEMP1".

Type 2 When there is no dot (.) in the dsname then the first 4 characters of the dsname must match your userid. The maximum length of the dsname is 8 characters. For example: "CCGWTMP1". Type 2 is available for compatibility with older versions of MUSIC.

## UDS Access Control

Access to a UDS is governed by the use of an indicator character included in the dsname. This character is in the 1st position after the dot (type 1) or in the 5th position (type 2). The indicator may be one of the following:

$ private - only the owner is allowed to read or write to it (Example: GEORGEW.$TEMP or CCGW$TMP).
# public - any user can write to it.
*x* read only - *x* represents any other letter or number and it means that any user can read but only the owner can write or delete.

Refer to the /FILE statement write up for more details.

## UDS Quotas

The installation may limit the maximum UDS file size that you can create.

*Note:* There are several utility programs available to help you with User Data Sets. See *Chapter 10. Utilities* for more information.

# Storing Data on a UDS File

Data can be copied from a file(s) to a UDS file using the program UTIL as described in the topic "UTIL" in *Chapter 10 - Utilities*.

If desired, both object modules and data may be stored on the same UDS file just so long as there is a /DATA control statement separating the object modules from the data. The program can read from the file using the I/O unit number given on the /FILE statement.

UDS files can be accessed using sequential or direct access input/output (I/O) statements. A UDS file created by one program using sequential I/O techniques can be used by another using direct access. A direct access data set written by one job, however, should not be read sequentially from another job as there will be no end-of-file marker after the last data record.

If you are careful, you can even use both the sequential and direct access techniques in one program. You cannot use both methods at the same time. Just do a rewind operation before starting to use the file in a direct access manner. Call the system subroutine CLOSDA before switching to a sequential access manner.

# Disk Block Utilization

MUSIC disk UDS files always use 512-byte blocks. MUSIC will often read and write them in groups, thus handling them as if they were 1024 through 10240 bytes in length. This multiple block read/write feature of MUSIC is performed automatically.

When a UDS file is created, you specify its record size. MUSIC uses this number together with the number of records you wanted in order to calculate how many 512-byte blocks to set aside for your file. For example, if your record size was 128 bytes, then it knows it can fit 4 per block. A record size of 80 bytes means that 6 will fit per block leaving 32 bytes unused.

# Buffer Allocation for UDS Files

Though buffer allocation for UDS files is automatic in MUSIC, the following explanation may be of assistance.

Each UDS file used in a job is allocated at least two buffers each. MUSIC may allocate 4 or even 6 buffers to a single UDS file. These additional buffers allocated automatically by MUSIC can considerably increase the efficiency of your program in some cases. It may be to the user's advantage to ensure that adequate space is left for buffers.

Buffers for UDS files on disk are always 512 bytes each for a total of 1024 bytes per pair of buffers. Each buffer for a UDS file on tape is equal to the blocksize (BLK) given on the /FILE statement defining it.

FORTRAN direct access data sets cannot use more than one pair of buffers per data set, although more may

be allocated for them.

## Available Buffer Space - UDS

The main storage available for buffers for UDS files (including files on magnetic tape) that may be used depends on which processor is being used and the size of the user region. The following descriptions are arranged by the name given on the /LOAD statement. If the specific compiler is not listed, then it probably is the same as /LOAD ASM.

ASM — Buffer space is whatever is required for tape and UDS buffers (2 buffers each), and is a minimum of 4096 bytes. A large enough user region must be used so that sufficient space remains for the program and GETMAIN area. Assuming a large enough user region, tape blocksizes may be up to 32760.

EXEC — Buffer space available is approximately the user region size, less 15K, less the length of the load module. The handling of buffers for tape files is described under FORTG1 processor given below.

FORTG1 — Buffer space available during the compile and loading of the program is 4096 bytes.

If the user specifies a /FILE statement for a magnetic tape file, then the system will attempt to allocate buffers for it when the job starts. If there is insufficient space at that time, it will try again once your program is loaded. The job will be terminated at this time if insufficient space remains.

Buffer space available during the execution phase is approximately the user region size, less 13K, less the size of the user program (not counting IBCOM). Buffers are reallocated just before the execution is to begin to take advantage of any extra space that may be available.

LKED — Buffer space during the Linkage Editor phase is the same as that described for /LOAD ASM. The Linkage Editor requires one UDS file or one file to hold the load module it produces. Unit number 4 is also used as a temporary UDS file during the link edit process.

Buffer space during the execution phase is described above under the EXEC processor.

LOADER — The buffer space for the LOADER is identical to that of FORTG1.

PLC — Buffer space available is 4096 bytes. The user may specify up to 3 UDS files.

PLI — Same as /LOAD ASM.

The user can use 3 UDS files. A UDS file on unit 4 contains the compiler modules and run-time transient library.

PLILG — Same as /LOAD ASM.

The user can use 3 UDS files. A UDS file on unit 4 contains the PL/I transient library modules, which are loaded dynamically during execution.

VSBASIC — Buffer space available is 4096 bytes. This space may be increased by using the BUF=n parameter on the first /OPT statement (refer to *Chapter 8. Processors* for the section on VS BASIC).

XMON — Same as /LOAD ASM.

XMPLI Same as /LOAD ASM.

The user can use 3 UDS files. A UDS file on unit 4 contains the PL/I transient library.

# MUSIC/SP Virtual Storage Access Method (VSAM)

## Introduction to VSAM

MUSIC/SP VSAM (Virtual Storage Access Method) is an implementation of the data organization method described in the IBM publication *OS/VS Virtual Storage Access Method* (*VSAM*) *Programmer's Guide*, GC26-3838. That publication describes the features and usage of VSAM from the point of view of an assembler programmer. Most of the assembler interface documented there is supported on MUSIC/SP for programs running in OS simulation mode. Some major features not supported by MUSIC/SP are: control interval access, spanned records, data set passwords.

The VSAM assembler interface is also used by higher level languages which allow access to VSAM files. In this way, MUSIC/SP programs written in languages such as PL/I, VS Cobol, and VS Fortran, in addition to VS Assembler, can use VSAM.

MUSIC/SP supports the three types of VSAM files: KSDS (key-sequenced data set), ESDS (entry-sequenced data set), and RRDS (relative record data set). The data records, which make up the *data component*, of a VSAM file are stored in a single file or UDS (user data set) file. In the case of a KSDS, the *index component* is stored in a second file. Alternate indexes (which are themselves KSDSs) can be built over a KSDS or ESDS. The Access Method Services (AMS) utility program is used to create and initialize VSAM files on MUSIC/SP.

The most significant feature of VSAM is its ability to access data records by means of a *key*, which is a fixed-length field at a fixed displacement in each data record. Each key, along with a pointer to the data record which contains the key, is also stored in a multi-level *index* (the primary index or an alternate index), which can be quickly searched for a given key. Instead of specifying a record number or an address on disk, the user program specifies the key when retrieving or updating a record. For example, a student number key could be used to identify a student record in a file of marks, or a part number could identify a part record within an inventory file. Records can be added, changed and deleted; VSAM automatically updates the index as required. It is also possible to access the data records sequentially, in order of increasing or decreasing keys.

## VSAM References

*OS/VS Virtual Storage Access Method* (*VSAM*) *Programmer's Guide*, GC26-3838.

*Introduction to IBM Direct-Access Storage Devices and Organization Methods* (student text), GC20-1649. Refer to the chapter on VSAM.

*OS/VS2 Access Method Services*, GC26-3841. On MUSIC/SP, a subset of Access Method Services is provided by the utility program AMS, which is described in *Chapter 10. Utilities*.

## VSAM Abbreviations

ACB — Access control block. An OS control block which is specified in order to open a VSAM file (logically connect a program to a VSAM file). The ACB is analogous to the DCB (data control block) for non-VSAM files.

| | |
|---|---|
| AMS | Access Method Services utility program. |
| CA | Control area. A logical group of control intervals. |
| CI | Control interval. A fixed-length area on disk, where VSAM stores data records. The CI corresponds to a logical block for non-VSAM files. |
| ESDS | Entry-sequenced data set. One of the 3 types of VSAM files. |
| KSDS | Key-sequenced data set. One of the 3 types of VSAM files. |
| RBA | Relative byte address. The displacement (expressed as a fullword binary integer) of a data record or control interval from the beginning of the data set. The VSAM RBA of the first control interval is 0. |
| RPL | Request parameter list. An OS control block which defines an input/output operation to be performed on a VSAM file. |
| RRDS | Relative record data set. One of the 3 types of VSAM files. |
| RRN | Relative record number. The record number (1, 2, 3,...) of a data record (slot) in an RRDS. |

# VSAM Data Storage and Organization

A KSDS consists of a data component and an index component. An ESDS or RRDS consists of a data component only (ignoring alternate indexes for the moment). The data component and index component each occupy one file on MUSIC. It is possible for the data component, but not the index component, to be a UDS (user data set) file instead of a regular file.

For example, a KSDS cluster could comprise the 2 files SAMPLE1.DAT (the data component) and SAMPLE1.IDX (the index component). Any file names can be used, but a naming convention such as this is recommended. The first 512-byte block of SAMPLE1.DAT contains the file name of the index, along with other control information. The user refers to the KSDS cluster by giving the file name of the data component, SAMPLE1.DAT, usually on a /FILE statement.

MUSIC/SP does not have a separate VSAM catalog. Attributes and control information for a VSAM file are stored in the first 512-byte block of the data component and index component files. User data starts in the second 512-byte block. For example, if the control interval size is 4096, which is eight 512-byte blocks, then the first data CI occupies blocks 2 through 9 of the file and is at VSAM RBA 0. The 2nd CI is blocks 10 through 17, at VSAM RBA 4096, etc. Control information is in block 1.

The VSAM RBA is different from the MUSIC/SP file system (MFIO) RBA. They are related by the formula: MFIO RBA = VSAM RBA + 1024.

An *alternate index* is a special type of KSDS which is associated with a KSDS or ESDS base cluster. It allows access to data records via a key field different from the primary key. Block 1 of the alternate index data component contains the file name of the data component of the base cluster. There can be several alternate indexes built over a KSDS or ESDS. The combination of an alternate index and a base cluster is called a *path*. Each path is given its own name and is represented by a 1-block file of that name. The path file contains the file name of the alternate index data component. So, when a base KSDS is accessed via a path, there are 5 MUSIC/SP files involved: the path file, the data and index components of the alternate index, and the data and index components of the base KSDS. The name of the path file is specified on the /FILE statement.

A MUSIC file which is part of a VSAM file normally has bit X'20' on in the first byte of the 4-byte access control field.

VSAM control intervals on MUSIC/SP have the same format as on OS, except that spanned records are not allowed.

VSAM indexes on MUSIC/SP have the same structure as on OS, and index records have the same basic format, except for the following differences. The first index record (record 0) starts at the 2nd 512-byte block of the file. Block 1 contains control information. There is only one *section* per index record. In the sequence set (lowest level), vertical pointers are always 3-byte CI numbers, relative to the beginning of the data component, which starts with CI 0. Sequence set records do not contain pointers to free CIs, since new CIs are always added after existing CIs in the data component. Sequence set records are chained backward as well as forward. A chain of free index records is maintained.

## Features Not Supported by VSAM

Compared with OS VSAM, the following restrictions apply to MUSIC/SP VSAM.

- Records may not span control intervals. This means that the longest record that can be stored is CILEN-7 bytes, where CILEN is the data component CI length. CILEN can be up to 32768 if necessary, but is fixed for a given file. This also restricts the number of pointers that can be stored in an alternate index record, for an alternate index with the NONUNIQUEKEY option. LRECL for an alternate index should be defined as CILEN-7 to allow for records as large as possible.

- Control interval access (MACRF=CNV and OPTCD=CNV) is not supported.

- Passwords for VSAM files are not supported.

- Reusable data sets (VSAM work files) are not supported. This is the REUSE option of the AMS DEFINE command.

- These performance options are not supported: IMBED, KEYRANGES, REPLICATE.

- The maximum number of alternate indexes in the upgrade set is 10. Note that, as on OS, performance is slow when updating a base cluster with several alternate indexes in its upgrade set, so the number of alternate indexes should be kept to a minimum.

- A data component is limited in size to the maximum file, which is usually 57000K, unless the data component is a UDS. However, an alternate index cannot be built over a UDS base.

- There are limitations on how VSAM files can be shared by multiple users and/or programs. In general, concurrent updating by multiple users or programs requires special handling. See the section on file sharing below.

- The OPEN/CLOSE message area is not supported. If a message area (MAREA and MLEN parameters) is defined for an ACB, it is ignored by MUSIC/SP.

- MUSIC/SP VSAM is available in OS simulation mode only. It cannot be used from Fortran G1 programs or other programs running in MUSIC (non-OS) mode.

- Only the following types of exit routines are supported: EODAD (end of data set), LERAD (logical error), SYNAD (physical error). An exit routine must not be specified as the member name of a module to be loaded dynamically by VSAM (the L option of EXLST).

- The following macros for advanced applications are not supported: BLDVRP, DLVRP, GETIX, MRKBFR, PUTIX, SCHBFR, SHOWCAT, VERIFY, WRTBFR.

- The following fields are not supported in SHOWCB and TESTCB macros for an ACB: AVSPAC, BFRFND, BUFNO, BUFRDS, ENDRBA, FS, HALCRBA, NCIS, NDELR, NEXCP, NEXT, NINSR, NIXL, NLOGR, NRETR, NSSS, NUIW, NUPDR, STMT, UIW.

# VSAM Usage Notes

A VSAM file to be used by a program must first be created and initialized by the DEFINE command of the AMS utility. At that time, you specify file characteristics such as name, type of file, amount of disk space required, data and index CI length, length (1 to 255) and position of the key field, and maximum logical record length. Refer to the description of the AMS utility for details.

For most applications, recommended index and data CI lengths are 512 and 4096, respectively. If the key length is 234 or more, the index CI length must be at least 1024.

In order to use the file in a program, a /FILE statement of the following form is required:

```
/FILE ddname NAME(filename) disp
```

ddname — is the 1- to 8-character data definition name used in the program. This name is specified in the ACB when the file is opened.

filename — is the name of the data component file. In the case of accessing a base cluster via an alternate index, it is the name of the path file.

disp — disp is the disposition. For read-only access, it should be SHR. For read/write access, it should be OLD NORLSE. The NORLSE option tells MUSIC/SP not to release unused space when the job ends. NORLSE can be omitted if you do not expect to make major additions to the file after this job. To allow concurrent access by multiple users or programs, some of which may be updating the file, the disposition should be WSHR NORLSE for read/write, otherwise SHR.

# Main Storage Requirements - VSAM

VSAM services during a user program are provided by the re-entrant modules VSAM (about 32K) and CBMANIP (about 14K). They are loaded from the system Load Library or Link Pack Area the first time the job requires them. They occupy main storage in the user region, unless your system administrator has placed them into the Link Pack Area. Module VSAM is always required for VSAM. CBMANIP is only required if any of the control block manipulation macros (GENCB, MODCB, SHOWCB, and TESTCB) are used. VSAM programs written in high-level languages such as PL/I require CBMANIP.

In addition to storage for modules, VSAM requires a work area of about 10K in the user region.

VSAM requires data buffers and index buffers (in the case of a KSDS) for each base cluster and alternate index cluster. They are allocated in the user region when the VSAM file is opened.

The length of each data buffer is the data CI length. There is 1 data buffer for each concurrent string (for most applications the number of concurrent strings is 1). An additional 1 or 2 work data buffers are required if a KSDS or alternate index base is being updated. These additional buffers are allocated automatically and should not be included in the number of buffers specified in the ACB or in the AMS DEFINE command.

The length of each index buffer is the index CI length, normally 512. There are no index buffers for an ESDS or RRDS. For a KSDS, there are at least 4 index buffers. Specifying more index buffers will usually increase performance, especially if the index is large and access is not sequential.

Each alternate index cluster is a KSDS, and therefore has its own set of data and index buffers.

Each open cluster requires approximately 2K additional storage for miscellaneous control blocks and work areas.

## VSAM File Sharing

A job that updates a VSAM file while other jobs are using the file must specify WSHR, rather than OLD, on the /FILE statement. Using OLD prevents all other jobs from accessing the file while the current job is running.

When WSHR is specified, it is not possible to add space to a file. Only the currently allocated space can be used.

When one job is updating a VSAM file while other jobs are accessing it for read-only, read integrity for the other jobs is the responsibility of those jobs. See the discussion of cross-region share option 2 in the VSAM Programmer's Guide. Also, it is up to the updating job to use ENQ or a similar facility to prevent other jobs from updating the file at the same time.

If multiple jobs are allowed to update a file concurrently, read and write integrity is the user's responsibility. See the discussion of cross-region share options 3 and 4 in the VSAM Programmer's Guide. In general, multiple concurrent updating is not recommended unless ENQ is used to serialize all requests and the ENDREQ macro (or CLOSE TYPE=T) is used to release buffers and positioning. MUSIC/SP VSAM does not prevent concurrent updaters from ignoring these requirements and damaging data.

The CLOSE TYPE=T request is useful when sharing files. It forces modified data buffers and block 1 buffers to disk, releases positioning, invalidates the main storage copies of index buffers, and rereads block 1 information from disk, but the data set remains open for further processing. It is logically similar to a CLOSE followed by an OPEN, but is more efficient. The ENDREQ macro does not invalidate index buffers or update block 1 information.

MUSIC/SP does not support the *shared resources* feature of OS VSAM. This refers to the sharing of control blocks and buffers among several VSAM data sets open at the same time.

## VSAM Error Codes

VSAM open, close, and input/output requests return an error code number if the request does not complete successfully. In some cases, the error codes for MUSIC/SP differ slightly from those for OS VSAM. The error codes for MUSIC/SP are listed below.

For open and close, the error code is returned in the byte at displacement 49 in the ACB. For I/O requests, the error code is in the 4th byte of the feedback word in the RPL. The feedback word is at displacement 12 in the RPL; the error code byte is at displacement 15. The register 15 (R15) return code for an I/O request is in the 2nd byte of the feedback word. An error code of 0 indicates successful completion.

**OPEN Error Codes When R15 Is Not 0**

Note: Always test for an error condition after an OPEN, since attempting to use an ACB which was not opened successfully usually causes a branch to location 0 (a program interrupt).

4 (Warning only.) The data set is already open.

128 The ddname is undefined, or an indicated file cannot be found.

144 An I/O error occurred while reading or writing the control information in block 1 of a VSAM file.

148 Unable to open a file, for a reason other than the conditions in OPEN error codes 128, 152, 168, and 172.

152 Requested access to a file is not allowed (MFIO error code 43).

160 Inconsistent or invalid control information in block 1.

168 File is in use (MFIO error code 33).

172 Too many open files (MFIO error code 20).

188 The block 1 identifier is incorrect, or the file is not a VSAM file, or a path file does not point to an alternate index, or an item in the upgrade set is not an alternate index.

**CLOSE Error Codes When R15 Is Not 0**

4 The data set is already closed.

144 An I/O error occurred while writing block 1 control information to disk, or (for CLOSE TYPE=T) while reading block 1.

148 Unable to close a MUSIC/SP file.

184 An I/O error occurred while VSAM was completing outstanding I/O requests.

**I/O Request Error Codes When R15=8 (Logical Error)**

(For error codes not shown here, refer to the VSAM Programmer's Guide.)

4 End of data set encountered. Either no EODAD routine is provided, or it returned to VSAM and the processing program issued another GET.

8 You attempted to store a record with a duplicate key, or there is a duplicate record for an alternate index with the unique key option.

16 Record not found.

32 You specified an RBA that is not the address of a data record.

44 For a GET with OPTCD=MVE, the receiving area is not large enough for the data record. The data is truncated and the GET completed normally except for this. For a GET with OPTCD=LOC, the receiving area is less than 4 bytes long.

64 The request cannot be started because the number of active requests would exceed the maximum number of strings.

68 Attempt to PUT or ERASE on a read-only file. The file may be read-only because SHR or WSHR was used on the /FILE statement, or because the ACB specified input processing only.

72 Keyed request on an ESDS is not allowed.

76 Insert by RBA is not allowed on a KSDS.

80 ERASE on an ESDS is not allowed.

84 OPTCD=LOC is not allowed for a PUT request or in an RPL in a chain of RPLs.

88 Required positioning does not exist, or you attempted an illegal switch between forward and backward processing. For example, you issued a sequential GET without having caused VSAM to be positioned for it.

92 PUT for update or ERASE is not preceded by a GET for update.

96 An attempt to change the prime key or key of reference when updating a record.

100 An attempt to change the length of a record when making an addressed (by RBA) update.

104 Invalid or conflicting RPL options.

108 Invalid record length for a PUT. It is negative, or 0, or larger than the maximum, or is too small to contain the entire key field, or is not equal to the slot size for an RRDS.

112 Key length is 0 or too large.

144 Invalid pointer (no associated base record) in an alternate index.

148 Unable to add a pointer to an alternate index record during upgrade processing, because the resulting length of the alternate index record would exceed the maximum record length or CILEN-7. The original PUT request is not done.

192 Invalid relative record number (RRN) for a request on an RRDS.

196 Addressed request is not allowed on an RRDS.

200 Addressed access is not allowed through a path.

204 PUT insert request is not allowed during backward processing.

**I/O Request Error Codes When R15=12 (Physical Error)**

4 Read error in a data component.

8 Read error in an index component.

16 Write error in a data component.

20 Write error in an index component.

**I/O Error Message Area Contents**

When a physical error occurs (return code R15=12 for an I/O request) and the RPL specifies a message area which is 128 bytes or longer, MUSIC/SP VSAM fills in the message area as described below. The first 12 bytes are the same as in OS VSAM. The remaining bytes are in printable format, and are different on MUSIC/SP and OS. Only the first 128 bytes are filled in. No bytes are filled in if the message area length is less than 128.

Bytes 0-1: Binary value 128 (message length).

Bytes 2-3: Unused (0).

Bytes 4-5: Binary value 124.

Bytes 6-7: Unused (0).

Bytes 8-11: I/O buffer address.

Bytes 12-127: Printable text in the format:

```
VSAM dddddddd t-ccccc I/O ERROR nnn: xx BLK=mmmmmm
LEN=llllll BUF=aaaaaa yyyyy...
```

Bytes 17-24: (dddddddd) The file's ddname.

Byte 26: (t) B for base cluster, A for alternate index cluster, or U for alternate index cluster being accessed during upgrade processing.

Bytes 28-32: (ccccc) DATA for an data component or INDEX for an index component.

Bytes 44-46: (nnn) MUSIC/SP file system (MFIO) error code.

Bytes 49-50: (xx) RD for a read operation or WR for a write operation.

Bytes 56-61: (mmmmmm) The starting 512-byte block number where the error occurred.

Bytes 67-71: (llllll) The length used in the MFIO I/O request.

Bytes 77-82: (aaaaaa) The buffer address used in the MFIO I/O request.

Bytes 84-127: (yyyyy...) The first 44 characters of the error description text corresponding to the MFIO error code. This describes the reason for the error.

# VSAM Abend Codes

When VSAM detects an unexpected error which it cannot recover from, it issues an abend (abnormal end) message on logical unit 6 (i.e. the workstation or whatever is defined as /FILE 6). The message contains an abend code number, which indicates the reason for the abend.

The format of the abend message is:

```
*VSAM ABEND nnn AT aaaaaa   INFO = xxxxxxxx xxxxxxxx xxxxxxxx
```

*nnn* is the abend code. *aaaaaa* is the address from which the abend routine was invoked. *xxxxx*.... gives 3

words of additional information, in hexadecimal. The meaning of the INFO words depends on the abend code.

Following the abend message, VSAM enters a conversational dump routine or dumps the user region (for a job running on batch). Then the job is terminated.

For a complete listing of the abend codes, list or edit file VSAM.ABENDS.

Some notable abend codes are:

5 Not enough index buffers.

6 Unexpected error while retrieving a data record from a control interval. This error usually indicates invalid control information in a data CI.

7 Attempt to access a spanned record. Spanned records are not supported.

10 Attempt to use control interval access. CI access is not supported.

14 Illegal change of request options in an RPL chain.

17 An invalid index record has been encountered. The first word of INFO has the index record number in hexadecimal.

19 Invalid record in the data component of an alternate index cluster.

22 Chained RPLs are not allowed with a PUT/update or ERASE request.

26 For a PUT request, the address of the logical record in the RPL (the RPL AREA field) is 0. I.e. no area address has been provided.

## Tracing and Debugging Facilities

Two options on the /JOB statement (or on the member name statement for /LOAD XMON) can be used to trace and debug the internal workings of the VSAM processing modules. In some cases these options are useful in determining why a user program is failing.

The IOTRACE option causes a trace message to be written to logical unit 6 for each entry to, or exit from, a VSAM processing routine. Since many routine calls are made for each user request, this option may generate a very large amount of output.

If the CDUMP option is used with IOTRACE, a conversational dump routine is entered after each VSAM trace message. Commands, read conversationally from logical unit 9, are used to inspect and modify main storage and registers, to skip a specified number of subsequent trace items, and to turn VSAM tracing on or off. Enter a blank line to resume execution. To suppress all further VSAM tracing, enter the command TRACE END and then a blank line (non-VSAM tracing will continue).

## VSAM Miscellaneous Notes

- For a GET request, if the address in the RPL of the receiving area for the logical record (the RPL AREA field) is zero, then no data is moved but the GET is completed normally and no error indication is given. This applies to both move and locate mode.

- It is possible to open a VSAM file dynamically. Normally a ddname is specified in the ACB when a file is opened. In that case the file must be defined by a /FILE statement. A special form of ddname is used to open a VSAM file which is not defined on a /FILE statement. This feature is available only on MUSIC/SP. The 8-byte ddname field is, in hexadecimal, 0Fxx0000 00aaaaaa, where 0F indicates this special type of OPEN, *xx* contains option bits, and *aaaaaa* is the address of a 22-character area which contains the full file name of the data component or path file. Bits in the option byte are: X'80' on means the cluster is to be opened read-only (equivalent to SHR on /FILE); X'40' bit is used internally by VSAM OPEN; X'20' on means that the cluster is to be opened for shared read/write (WSHR on /FILE). Option byte X'00' opens the cluster for non-shared read/write (OLD on /FILE).

- The OS RDJFCB macro (read Job File Control Block, SVC 64) can be used before OPEN to test whether a ddname represents a VSAM or non-VSAM file. This is useful to know, since opening a VSAM file requires an ACB while opening a non-VSAM file requires a DCB. On MUSIC/SP, the byte at displacement 99 (JFCDSRG2) in the JFCB for a VSAM file has bit X'08' on. After the RDFJCB, do a normal OPEN, not OPEN TYPE=J. Refer to RDJFCB in *OS/VS2 System Programming Library*: *Data Management*, GC26-3830.

# PL/I Support

The environment option REUSE is not supported by MUSIC/SP.

The environment option BUFND(n) should not be used, since extra data component buffers will not improve performance. The BUFSP(n) option is not supported. Use BUFNI(n) to specify the number of index buffers if you wish to use more than the standard number.

The VSAM Compatibility Interface, which allows a VSAM file to be accessed by a PL/I program originally written to access an ISAM file, is not supported.

# Sample Program for VSAM

The sample assembler program presented here uses the macros described in the VSAM Programmer's Guide to copy records from a sequential file, MYDATA, to a VSAM KSDS file, VSAMFILE1.DAT. The records are added as new records.

The following job creates and initializes the data and index components of the KSDS using the AMS utility. The key field occupies the first 8 bytes of each record. The initial space allocation for the data component is 20K. The data control interval size is 4096. The name of the index component file is VSAMFILE1.IDX. The index control interval size is 512 by default.

```
/INCLUDE AMS
DEFINE CLUSTER(NAME(VSAMFILE1.DAT) INDEXED SPACE(20) -
  CISZ(4096) KEYS(8 0))
```

The sample program is listed below. It is contained in file VSAM.SAMPLE.

```
/FILE DDNAME1 NAME(MYDATA) SHR
/FILE DDNAME2 NAME(VSAMFILE1.DAT) OLD NORLSE
/LOAD ASM
*
* THIS SAMPLE PROGRAM READS DATA RECORDS FROM A SEQUENTIAL
* FILE ON DDNAME "DDNAME1" AND WRITES THEM TO A VSAM KSDS
* ON DDNAME "DDNAME2".  IT IS ASSUMED THAT THE VSAM FILE
```

```
* HAS ALREADY BEEN CREATED AND INITIALIZED.
* THE INPUT RECORDS ARE ASSUMED TO BE OF LENGTH 80.
*
* AN OPEN ERROR CAUSES AN INVALID OP-CODE WITH R7=X'EE1' AND
* R0=OPEN ERROR CODE.
*
* AN ERROR WRITING TO THE VSAM FILE CAUSES AN INVALID OP-CODE
* WITH R7=X'EE2', R0=ERROR CODE FROM PUT, R15=RETURN CODE FROM PUT.
*
SAMPLE   CSECT
         REGS  ,                  THIS MACRO DEFINES REGISTERS
         STM   R14,R12,12(R13)    SAVE REGISTERS
         LR    R12,R15            SET UP A BASE REGISTER
         USING SAMPLE,R12
         LA    R14,SAVEAREA       SET UP A SAVE AREA
         ST    R13,4(0,R14)
         ST    R14,8(0,R13)
         LR    R13,R14
* OPEN THE TWO FILES
         OPEN  (MYDCB,INPUT)      OPEN SEQUENTIAL FILE
         OPEN  (MYACB)            OPEN VSAM FILE, USING AN ACB
* GET ERROR CODE FIELD FROM ACB
         SHOWCB ACB=MYACB,AREA=OPENERRC,LENGTH=4,FIELDS=(ERROR)
         L     R0,OPENERRC
         LTR   R0,R0              TEST FOR VSAM OPEN ERROR
         BZ    OPENOK             BRANCH IF NO ERROR
* VSAM OPEN ERROR
         LA    R7,X'EE1'          INDICATE INTENTIONAL P.I.
         DC    H'0'               STOP JOB BY INVALID OP-CODE
OPENOK   DS    0H
* READ LOOP: READ NEXT RECORD INTO "MYAREA"
READLOOP GET   MYDCB,MYAREA       READ RECORD USING QSAM
         PUT   RPL=MYRPL          WRITE RECORD TO VSAM FILE
         LTR   R15,R15            TEST FOR WRITE ERROR
         BZ    READLOOP           BRANCH IF NO ERROR
* VSAM ERROR: GET ERROR CODE FROM RPL
         LR    R5,R15             SAVE R15 RETURN CODE
         SHOWCB RPL=MYRPL,AREA=REQERR,LENGTH=4,FIELDS=(FDBK)
         L     R0,REQERR          GET ERROR CODE IN R0
         LR    R15,R5             RESTORE R15 RETURN CODE
         LA    R7,X'EE2'          INDICATE INTENTIONAL P.I.
         DC    H'0'               STOP JOB BY INVALID OP-CODE
* COME HERE WHEN END-OF-FILE ON INPUT: CLOSE THE FILES
EOF      CLOSE (MYDCB)
         CLOSE (MYACB)
* WRITE MESSAGE TO TERMINAL
         WTO   'VSAM SAMPLE PROGRAM ENDED NORMALLY'
* RETURN TO SYSTEM
         L     R13,4(0,R13)       RESTORE SAVE AREA POINTER
         LM    R14,R12,12(R13)    RESTORE REGISTERS
         BR    R14                RETURN
* STORAGE AREAS
SAVEAREA DS    18F                STANDARD SAVE AREA
OPENERRC DS    F                  RECEIVES OPEN ERROR CODE
REQERR   DS    F                  RECEIVES ERROR CODE FROM RPL
MYAREA   DS    CL80               LOGICAL RECORD BUFFER
```

```
* DCB FOR SEQUENTIAL INPUT FILE
MYDCB    DCB   DDNAME=DDNAME1,DSORG=PS,MACRF=GM,LRECL=80,EODAD=EOF
* ACB FOR OPENING VSAM FILE
MYACB    ACB   DDNAME=DDNAME2,MACRF=(KEY,SEQ,OUT)
* RPL FOR REQUESTS ON THE VSAM FILE
MYRPL    RPL   ACB=MYACB,AREA=MYAREA,AREALEN=80,RECLEN=80,        X
               OPTCD=(KEY,SEQ,NUP)
         END
```

MUSIC/SP User's Reference Guide

Part 2 - Chapters 5 and 6 (UR_P2.PS)

# Chapter 5. MUSIC Commands

# MUSIC Commands - Overview

MUSIC commands cause the specified action to occur immediately. (They are sometimes referred to as immediate commands.) Most commands are entered in command mode or from the command line of many programs. The workstation is in Command Mode when the *Go message is displayed. Variable parameters are separated by blanks or commas as indicated in the command description.

If a command is entered in Break mode, the "/" is required. The slash may be required in front of a MUSIC command when it is entered from the command line of a currently running program. The slash distinguishes MUSIC commands from commands valid for the current program. For example, while using the Editor program, use "/COPY" for the MUSIC command (COPY is also an Editor command).

A large number of MUSIC commands have meaning only when used from a workstation. Consult *Chapter 3. Using Batch* for a list of commands that may be used from batch.

Many MUSIC commands can be combined in command procedures as described under the REXX Command Executor topic in *Chapter 8. Processors*. REXX can be used to effectively tailor MUSIC commands and Job Control Statements to suit each user. (See *Chapter 6. Job Control Statements* for more information.)

## Workstation Modes

Some commands are valid only at certain times. For example, a command to cancel a job is only valid when the job is running. The various possible modes are defined below.

Command Mode — The workstation is in this mode when the *Go message appears. After you have signed on MUSIC, you are normally in this mode. From this mode you can start running a job, list a file, etc. This is also known as *Go mode.*

Execution Mode — The workstation is in this mode when MUSIC is running a job for you. Your workstation may display the message `*In progress` as it goes from command mode to execution mode.

Reading Mode — A program that is running may ask for some input from the workstation. At this point the workstation is in reading mode waiting for you to respond to the read. The job does not continue until after you enter the required information. Once you have entered the required line, your workstation goes back into execution mode. You may type /CANCEL when your workstation is in reading mode to cause your job to be terminated.

Break Mode — (This is also called attention mode.) The workstation enters this mode when you press the BREAK or ATTN key on your workstation (PA1 on a 3270-type workstation) and the system responds by allowing you to enter a break mode command. At this time you can enter a /CANCEL command to stop your job and return to *Go mode, or one of many other commands allowed in break mode, in which case, your workstation returns to execution mode. A preceding slash "/" is required for a

---

* Some users may not go directly to command mode, but instead, into a subsystem program such as FSI or TODO. In this case, MUSIC commands can be entered from the command line on the screen.

command entered during break mode. On a 3270-type workstation in break mode (indicated by `**Attn**` in the lower right corner), you can press the PA1 key to skip the remaining output. MUSIC does not stop your job just because your workstation has gone into break mode.

# Functional Summary of Commands

The following commands are divided into groups by function, each with a brief description. Details about each command is given later.

## Accessing Menu Facilities

| | |
|---|---|
| CI | Invokes the Class Information facility for students. |
| FSI | Invokes the FSI (Full Screen Interface) subsystem for general users of MUSIC. |
| PROG | Invokes the PROG (Programmer's Menu) subsystem for student users. |
| CM | Invokes the Instructor's Course Management Facility. |
| TODO | Invokes the TODO (Time, Office, and Documentation Organizer) subsystem for electronic office functions. |

## Editing

| | |
|---|---|
| BASIC | Edits a VS BASIC program. |
| DW370 | Invokes the optional IBM DisplayWrite/370 word processing product. |
| EDIT | Invokes the Editor. |
| BIGEDIT | Invokes the Editor with a large work file. |

## Controlling Workstation Input/Output

| | |
|---|---|
| /CANCEL | Terminates whatever activity is in progress. |
| /COMPRESS | Indicates whether or not to delete leading blanks and to compress multiple blanks to one blank in subsequent output lines. |
| /DEFINE | Defines 3270 program function keys. |
| /RECORD | Controls the writing of session recording file. |
| /SKIP | Skips all or part of output on the workstation. |
| /WINDOW | Sets the range of columns to be displayed on the workstation for the subsequent output lines until the *Go message appears. |

## Manipulating Files

| | |
|---|---|
| AMS | Invokes AMS (Access Method Services) that supports VSAM. |
| ATTRIB | Displays the attributes of the file specified. |
| BROWSE | Similar to EDIT except that you are in Read Only (RO) mode. |
| COMPARE | Compares the lines in two different files and reports the differences. |
| COPY | Copies one file into another, using the same attributes, while leaving the original file unchanged. |
| COUNT | Displays the usage counts and creation dates for files. |
| DECRYPT | Decodes a file that has been encrypted by the ENCRYPT program. |
| DISPLAY | Displays all or part of a file at the workstation. Each line displayed is preceded by a line number. |
| ENCRYPT | Encrypts (codes) files for added security. |
| EVIEW | Invokes the VIEW program for viewing encrypted files. |
| FINDTEXT | Searches some or all of your files for a specified string. |
| FLIB | Lists the names of your files and directories. This program is one of the selections of FSI. |
| LIBRARY | Lists names of all files currently owned by the user. |
| LIST | Same as DISPLAY, except line numbers are not shown. |
| OUTPUT | Invoke OUTPUT program to inspect batch output files. |
| PRINT | Print a file on the system or auxiliary printer. |
| PURGE | Permanently removes (deletes) a file (aliases: DELETE, ERASE) |
| RENAME | Renames a file. |
| SORT | Sorts the records of a file. |
| SUBMIT | Submit a file to be executed by MUSIC batch or some other batch processor. |
| SUMMARY | Displays a brief summary of the contents of a file. |
| TAG | Displays or changes the tag information associated with a file. |
| VIEW | Invokes the VIEW program for viewing files of any record length, type, or size. |
| ZEROCNT | Resets the usage count for a file to 0 by archiving it to a temporary file, then restoring it. |

## Changing File Attributes

| | |
|---|---|
| CHMOD | changes file attributes for a single file or a list of files. |
| MAKAPPO | changes file attribute to append only. |
| MAKCOM | includes file in the common index. |
| MAKPRIV | changes file attribute to private. |
| MAKPUBL | changes file attribute to public. |
| MAKSHR | changes file attribute to share. |

## Directories

| | |
|---|---|
| CD | Changes directories. |
| DIR | Lists file names for directories. |
| MD | Makes directories. |
| RD | Removes directories |
| TREE | Provides a graphical display of your directories. |

## Displaying Information

| | |
|---|---|
| CONF | Invokes the conference facility. |
| CONFMAN | Invokes the CONFMAN program for setting up and administering a conference. |
| IDP | Invokes the Information Display Program for creating help facilities and bulletin boards. |
| MAN | Invokes the word search facility for viewing MUSIC/SP manuals. |

## Signing On and Off

| | |
|---|---|
| /DISCON | Disconnects the workstation from an active session and leaves a program running. |
| /ID | Signs on to MUSIC. |
| OFF | Signs off from MUSIC. |

## Getting System Information

| | |
|---|---|
| HELP | Invokes the HELP program that provides help on a wide variety of topics. |
| NEWS | Lists current news items. |
| SESSIONS | Displays information about your multi-sessions. |

| | |
|---|---|
| /STATUS | Displays information about the user's workstation and the system. |
| SYSDATE | Displays the current time and date. |
| /TIME | Tells time of day and number of service units used for current job running up to that point. |
| /USERS | Displays number of users signed on to MUSIC. |
| VER | Displays current version of MUSIC. |
| WHOAMI | Displays information about your userid, TCB, ownership id, etc. |

## Changing Userid Attributes

| | |
|---|---|
| NEWPW | Changes your MUSIC sign-on password. |
| PROFILE | Invokes the PROFILE program for displaying or changing userid attributes, passwords, and limits. |

## Internet Access (TCP/IP)

| | |
|---|---|
| FINGER | Sends a one-line query to a remote Internet site. |
| FTP | Invokes the File Transfer Protocol program for transferring files between sites that are connected via TCP/IP to the Internet. |
| GOPHER | Connects to a GOPHER server for document search and retrieval. |
| IRC | IRC (Internet Relay Chat) allows you to carry on conversations with groups of people world-wide. |
| NET | Displays a list of network nodes and allows you to make a connection using FTP, GOPHER, or TELNET. |
| PING | The PING (Packet Internet Groper) command measures round-trip-times to internet sites. |
| PLAN | Invokes the Editor for updating your @PLAN file. |
| QFTP | Invokes Quick FTP for transferring files. |
| RN | Invokes the News Reader for Usenet. |
| TELNET | Allows you to establish sessions with computers on the TCP/IP Internet. |
| WEB | Invokes the line-mode browser for accessing Internet Web sites. |

## Sending Messages

| | |
|---|---|
| CHAT | Invokes the CHAT facility for interactive conferencing. |
| GETMAIL | Invokes the GETMAIL utility to read your incoming mail and store the text in a file. |
| GETMINFO | Invokes the GETMINFO utility to list information about your incoming mail items and |

| | |
|---|---|
| | stores this in a file. |
| LM | LM (List Manager) helps you manage your discussion lists. |
| MAIL | Invokes the MAIL program for sending electronic mail. |
| MESSAGES | Control reception of messages. |
| REQUEST | Sends a one-line message to the console operator. |
| SENDFILE | Sends a file of any record length to MUSIC or CMS. |
| SENDMAIL | Invokes the SENDMAIL utility for a fast-track method to send a mail item. |
| TELL | Send a one-line message to a user. |

## Controlling Multi-Sessions

The following commands are available for 3270-type workstations only.

| | |
|---|---|
| ADD | Signs the current workstation to another session. |
| DELETE | Deletes the current session. The DELETE command does not work if this is your only session. If the file name parameter is added, then it deletes files. |
| MS | Displays information about each session on your workstation. |
| NEXT | Goes forward to the next session in the chain. |
| PREVIOUS | Goes backward to the previous session in the chain. |
| SESSIONS | Displays information about your multi-sessions. |

## Graphics

| | |
|---|---|
| GDDM | Invokes the optional IBM GDDM graphics utilities. |

## Personal Computer WorkStation

| | |
|---|---|
| XTPC | Using the PCWS program, transfers files to the PC from MUSIC. |
| XTMUS | Using the PCWS program, transfers files to MUSIC from the PC. |
| PCEXEC | Using the PCWS program, starts up a PC application from MUSIC. |

## Miscellaneous Commands

| | |
|---|---|
| ACCESS | Invokes the MUSIC Passthru interface to access other systems. |
| EXECUTE | Executes program contained in a file and passes parameters to the program when it is running. |

| | |
|---|---|
| CICS | Invokes the MUSIC interface to the optional IBM CICS product. |
| COBTEST | Invokes the interactive debugger for the optional IBM VS COBOL compiler. |
| DEBUG | Invokes the DEBUG utility for debugging programs at the machine language level. |
| LANG | Set National language option. |
| MEET | Invokes the program from the TODO facility that schedules meeting rooms. |
| PHONE | Invokes the program from the TODO facility that maintains a log of telephone conversations. |
| PIPE | Invokes a device-independent pipe-line interface under MUSIC. |
| POST | Submits an article to one or more newsgroups of USENET. |
| RN | Invokes the News Reader for exchanging messages on USENET. |
| ROUTE | Displays or changes the current route destination. |
| SCHED | Invokes the TODO scheduling program for processing personal calendars, rooms, or equipment. |
| SHOPAN | Displays a panel file without changing it. |
| TUT | Invokes the Tutorials facility for learning programming languages. |

# MUSIC Commands Equivalent to Other Systems

MUSIC supports some commands from other systems (UNIX, DOS, or CMS). If the command you try is identical to a MUSIC command then that command is invoked. If MUSIC has a similar command to the one you enter then you will receive a message suggesting the appropriate MUSIC command.

Below is a short list of commands for UNIX that are similar to MUSIC commands. All the commands below can be typed an the MUSIC prompt (*Go) or on the command line of most applications.

Do not forget that MUSIC offers a Full Screen Interface (FSI) facility where the following commands (as well as many other MUSIC commands and utilities) can be easily accessed through menus. To invoke FSI, type "FSI" or "MENU" at the *Go prompt.

| *UNIX* | *MUSIC* | *Comments* |
|---|---|---|
| help | help | |
| man | man | this command accesses MUSIC manuals for text searching. It is organized by publication and not sections. |
| ls<br>ls - a | lib * p | MUSIC does not have hidden files. It creates and uses several files that start with the "@" character. |

| | | |
|---|---|---|
| dir | dir or lib | FLIB is a full screen library directory. |
| vi | edit | The MUSIC editor is similar to another Unix editor called pico. |
| cd | cd | |
| mkdir | md | |
| rmdir | rd | |
| passwd | newpw | |
| cp | copy | |
| rm | delete or purge | |
| mv | rename | |
| lp / lpr | print | |
| pwd | <ENTER> | |
| elm | mail | |
| grep | findtext | |
| cmp | compare | |
| cat | list | |
| logout/<CTRL-D> | off | |

## File Attributes

The following shows MUSIC equivalents to UNIX for accessing files:

| *UNIX* | *MUSIC* | *Effect* chmod |
|---|---|---|
| ugo+/-[rwx] | makpriv* | u+[rwx] g-[rwx] o-[rwx] |
| | makshr ** | u+[rwx] g+[rwx] o+[rx] |
| | makpubl | u+[rwx] g+[rx] o+[rx] |
| | makxo | u+[rwx] g+[x] o+[x] |

* all files are PRIV (private) as default.
** to rwx, g and o must know full filename (e.g. Owner:FileName)

# MUSIC Command Descriptions

## Command Presentation

The remainder of this chapter gives a description of all commands in alphabetic order.

Job Control Statements (sometimes referred to as *deferred* commands) can be found in the next Chapter.

### Conventions

The following conventions are used in this manual to describe the commands:

1. Upper case letters and punctuation marks represent information that must be coded exactly as shown.

2. Lower case letters and words are generic terms representing information that must be supplied. That is, a substitution must be made when coding a parameter or option so represented.

3. Information within square brackets [ ] represents an option or choice of options that may be included or omitted, depending on the requirements.

4. In the examples, information typed by the user is shown in a larger and bolder print than the information shown by the computer.

5. Underlined parameters are the default values.

6. In most cases and for most commands the preceding / is optional. The / is shown in the command description if it is necessary. For example, if a command is valid during Break mode and Command mode, the / is shown even though it is not required during Command mode. During Break mode the / is always required. The /CANCEL command requires the / at all times.

   A slash preceding a command can be used to distinguish MUSIC commands from commands for the current program. For example, if you wanted to use MUSIC's COPY command while in the Editor program, type "/COPY" (otherwise you will get the Editor's COPY command).

7. Many MUSIC commands are used with a file name. For example, "EDIT filename'. If the file name is not specified the *Input File* (@INPUT) for your userid is assumed.

### Abbreviations

Most commands have abbreviations which may be used only from workstations. These abbreviations, shown in the description of each command, represent the shortest form of the command. Any abbreviation between the shortest form and the full form is valid.

# ACCESS

This facility allows you to access systems outside of MUSIC such as CMS, VSE, MVS, and other MUSIC systems using the MUSIC Passthru interface.

**Syntax:**

```
ACCESS accessname [parameter]
ACCESS ?
ACCESS
```

**Parameters:**

accessname is the name of the external facility that you wish to access.

parameter is an optional value that is passed to the external facility once it is started.

? displays the list of external facilities available at your site.

Brief help is displayed when ACCESS is entered without a parameter.

*Note:* Your site may not allow/support this facility.

**Examples:**

```
access telnet nic.ddn.mil
```

logs in to foreign host nic.ddn.mil using the VM/SP telnet program, a TCP/IP protocol.

```
access sql
```

accesses sql on the local VM/SP system.

# ADD

This command is used to create extra MUSIC sessions. New sessions are added in a chain that can be accessed using the PREVIOUS (F7) and NEXT (F8) commands. Sessions are deleted with the DELETE (F5) command.

By default the F4 key has the ADD command as its definition for *Go mode. To add another session while using a full screen program like the Editor, press PA2 before the F4 key. See *Chapter 2. Workstations* under the heading "Multi-Session Support" for more information.

**Syntax:**

```
/ADD
```

# AMS

This command invokes the MUSIC/SP Access Method Services (AMS) utility program. This program supports VSAM (Virtual Storage Access Method). AMS allocates, manipulates, and generally maintains VSAM datasets. See *Chapter 10. Utilities* for more complete information.

**Syntax:**

```
AMS
```

# ATTRIB

This command displays the attributes of a specified file. These include the creation date, date last read, the number of lines, and its record length.

**Syntax:**

```
ATTRIB  filename
AT
```

**Parameters:**

filename is the name of the file.

**Example:**

```
*Go
attrib work1
*In progress
NAME=CCFP:WORK1           PRIV
TAG=
LRECL=80   RECFM=0200 (FC)  ACCESS CONTROL=00 00 C0 C0
SPACE (K):  PRIMARY=2  SECONDARY=0   MAXIMUM=-1
LINES=14    HIGH BLK=1  (MAX=3)   EOF DISPL=268
BACKUP NUMBER=147     USAGE COUNT=0    EXTENTS=1
CREATED 12MAR85  LAST READ 000000  LAST WRITTEN 12MAR85
CREATOR: CCFP000    LAST WRITER: CCFP000
*End
*Go
```

# BASIC

This command invokes the BASIC Editor. This Editor is used to edit a VS BASIC programs. Use the special name of NEW to tell the Editor that you wish to start entering a new program. The commands that you use with this Editor and examples of its use are located in *Chapter 8. Processors* of this guide.

**Syntax:**

```
BASIC  [filename]
BA
```

**Parameters:**

filename is the name of the file. If a name is not specified then the Input File is edited.

# BIGEDIT

This command is the same as the EDIT command, except that is uses a larger editor work file. This allows a larger file to be edited, up to about 12000 80-byte records.

# BROWSE

This command allows you to browse the contents of a file. BROWSE is equivalent to the EDIT command with the RO option and the CASE IGNORE Editor command in effect. For example, "BROWSE filename" is the same as "EDIT filename RO;CASE I". See the EDIT command for more information.

**Syntax:**

```
BROWSE [filename] [LRECL(n)] [UIO] [MAX(n)]
B                 [n       ]
```

**Parameters:**

filename is the name of the file.

LRECL(n)
n is the record length to be used during the session.

UIO causes the Editor to read the file by 512-byte records.

MAX(n) specifies the maximum number of records the Editor is to read from the file.

# /CANCEL

This command instructs the system to terminate whatever activity is in progress as well as stop all output associated with it. The workstation returns to *Go mode. See the notes below. If your workstation is in execution mode you must first press the ATTN or BREAK key to force the workstation to break mode before typing in this command.

You may type this command as response to any read that your job may have done.

**Syntax:**

```
/CANCEL [ALL]
/CA
```

**Parameters:**

ALL indicates that any always-program is also to be cancelled and could be used, for example, to terminate a chain of programs scheduled from REXX.

**Example:**

```
*Go
exec hello
*In progress
          (ATTN or BREAK key pressed)
/cancel
*Terminated
*Go
```

*Notes:*

1. When you are using a menu driven program such as TODO, entering the /CANCEL command to a conversational read returns you to a previous menu. /CANCEL ALL returns you directly to MUSIC command (*Go) mode.

2. Some programs may be set up as non-cancellable in which case the /CANCEL command will be rejected. This non-cancellable feature is in effect if your userid is set up to automatically run a job when you sign on.

# CD

This command is used to change your current directory to the one specified in the *dirname* parameter.

**Syntax:**

```
CD [dirname]
CD..
CD\
```

If *dirname* is not specified, then the name of your current directory is displayed.

Directories contain names of files that you want to keep together as a group. Directories are created by specifying their name on a MD (Make Directory) command. The CD command is used to set the name of your current directory. New files that you create will be stored by default in your current directory. The system will search the current directory first when it looks for an existing file.

When you first connect to MUSIC, you are in the "root directory". You can return to the root directory by

issuing the "CD \" command.

Directories can be established within directories. For example, you can have a directory called PARTS within a directory called CAR. You can use the command "CD \CAR\PARTS" to set the current directory to the one for PARTS that is under the CAR directory. Alternately you can issue the two commands "CD \CAR" followed by "CD PARTS" to do the same thing. Notice that "CD PARTS" was not written "CD \PARTS". That would have tried to find a directory called PARTS in the root directory. But since PARTS was in the CAR directory it would not have found it.

When you are at the root directory, specifying a command like "CD CAR" or "CD \CAR" have the same affect. When you are not at the root directory, the two commands will act differently as explained above.

Use the TREE command to visually show the names of your directories and how they relate to each other.

You can add the characters "..\" in front of parm to mean the same as the name of the "parent" directory. Suppose you are in the "\CAR\PARTS" directory and want to get to the "\CAR\SALES" directory. You can do this by using the command "CD ..\SALES" command.

**Examples:**

```
*Go
md car

*Go
cd car

*Go \CAR>
md parts

*Go \CAR>
cd parts

*Go \CAR\PARTS>
```

# CHAT

The CHAT command brings up the CHAT screen. A chat signal "*CHAT" is sent, via the TELL command, to each user you want to chat with. Anything you type in the command area is sent to the specified users (if any), via the TELL command, and appears on the recipients' CHAT screens, if they have started CHAT (if not, they get the message in the usual box at the top of the screen). Exceptions: (1) Text in the command area starting with / is assumed to be a MUSIC command and is executed. (2) A CHAT command ("chat userid") can be entered in the command area to add a user to the list of people you want to chat with; this form of CHAT command takes only 1 userid.

When someone exits from CHAT, an end signal ("*END") is sent to you. That removes the user from your list.

**Syntax:**

```
CHAT [userid1] [userid2] ... [-NOLOG]
```

Outgoing and incoming messages, prefixed by the sender's userid, are appended to log file *USR:@CHAT.LOG, unless -NOLOG option is used.

# CHMOD

This command changes the file attributes for a single file or a list of files. When a MUSIC file is created, it is given various attributes that control how the file can be accessed by you (the owner) and other users. The commands CHMOD ("change mode") and MAKxxxx (where xxxx has various values) are used to change the access attributes of an existing file.

You can use the ATTRIB command to display information about a file, including access, tag, count, and dates. E.g. attrib myfile For info about file usage counts, see the COUNT command.

Normally the owner of a file has unlimited access (unless the file is execute-only), while other users have limited access. The following keywords are used to refer to access attributes:

PRIV The file is private. Only the owner can access it.

PUBL The file is public. Non-owners can read the file, but not change it. The file name is in the common index, so that non-owners can refer to it without specifying the owner's userid.

SHR The file is "sharable". Non-owners can read the file, but not change it. The file name is NOT in the common index; a non-owner must specify the owner's userid to access it. For example, if user ABCD creates file DATA1 as SHR, then user EFGH must use the full file name ABCD:DATA1 to read the file. SHR is indicated in the output of the ATTRIB command by "RD NOWR" (read, no write).

COM The file name is in the common index (see PUBL). If COM is used alone, the file is normally private.

XO The file is execute-only. It can be executed as a command, but not otherwise read or changed.

APPO The file is append-only. Data records can be added to the end of the file. This type of access is usually used for log files.

LOG Same as APPO.

WRITE The file is writable by non-owners.

CNT The system maintains a usage counter for the file.

**Syntax:**

```
CHMOD filespec attrib     or    CHMOD <file attrib
```

**Parameters:**

filespec is a file name specification that may include wildcard characters

<file specifies a file containing a list of file names (one file name per record, with the name starting in column 1). Each of the files in the list will be changed to the specified

attribute(s).

| | |
|---|---|
| attrib | PRIV or PRIV,XO<br>SHR or SHR,XO<br>COM or COM,XO<br>PUBL or PUBL,XO<br>LOG or COM,LOG |

**Example:**

```
chmod myfile shr
```

See also: MAKxxxx commands.

# CI

This command invokes the Class Information facility for students. This menu program works in conjunction with CM (Course Management facility) for instructors. Help is available once the program is invoked.

**Syntax:**

```
CI
```

# CICS

This command is used to invoke the optional IBM CICS product. For details see *Chapter 8. Processors*.

**Syntax:**

```
CICS
```

# CM

This command invokes the Course Management facility for teachers. For more information refer to the *MUSIC/SP Teacher's Guide* or type "HELP CM".

**Syntax:**

```
CM
```

# COBTEST

This command is used to invoke the interactive debugger for the optional IBM VS COBOL compiler. For details see *Chapter 8. Processors*.

**Syntax:**

```
COBTEST
```

# COMPARE

The COMPARE command compares the records in two sequential files and reports records which are different and records which are in one file but not the other.

**Syntax:**

```
COMPARE filename1 filename2 <n-m> <ENDCHK> <TBL(k)>
COMP                        <m  >
```

**Parameters:**

filename1 The name of the first file. This file is called the "A" file.

filename2 The name of the second file. This file is called the "B" file.

n-m Starting (n) and ending (m) column numbers of the area to be compared in each record. The default is *n* = 1 and *m* = the larger of the two record lengths, to a maximum of 512 (i.e. for files with record length 512 or less, all characters of each record are compared). For purposes of comparison, a shorter record is considered to be extended with blanks.

When only one column number is specified, it is the ending column, and the starting column is assumed to be 1.

ENDCHK (Optional) This option causes object module END records to be compared like any other records. Without this option, object END records are compared only on columns 1-16 (provided the compare area includes 1-16).

TBL(k) *k* is a number from 2 to 120. This option specifies how many records to search ahead when unequal records are found. The command searches forward in each file for a matching record; if found, the intervening records are reported as records in one file but not the other. The default is TBL(120). You can specify a smaller number to restrict the look-ahead.

**Output:** Unequal and mismatched records are displayed with their line numbers and a flag "A" (records in file 1 but not in file 2), "B" (in file 2 but not in file 1), or "AB" (unequal records).

**Return Codes:**

| | |
|---|---|
| 0 | Files compare equal in the specified column range. |
| 1 | Files compare unequal. |
| 8 | An error occurred opening one of the files. |
| 12 | An error occurred reading one of the files. |
| 16 | An invalid command option was specified. |

**Examples:**

```
comp data1 data2
```
Compares files DATA1 and DATA2 (all of each record, to a maximum of 512 bytes per record).

```
comp data1 data2 11-30
```
Compares columns 11 through 30 of files DATA1 and DATA2.

```
comp progx.obj progy.obj 72 endchk tbl(10)
```
Compare columns 1-72 of two object modules, with full compare of object END records, and look-ahead limited to 10 records.

# /COMPRESS

This command, when used in break mode, causes all succeeding output lines to be compressed. All multiple blanks appearing in each line are compressed to one blank. All leading blanks are also deleted. This option is reset the next time your workstation enters command mode.

**Syntax:**

```
/COMPRESS [ON ]
/COM      [OFF]
```

**Parameters:**

ON is the default parameter to turn on compression.

OFF cancels the effect of this command.

**Example:**

```
*Go
list sample
*In progress
1      2      3      4      5
1      2      3      4(ATTN or BREAK key pressed)
/compress
*OK
6 7 8 9 10
11 12 13 14 15
*End
*Go
```

*Note:* This function reduces output time, and can make output more readable, particularly if lines longer

than 72 characters in length are to be output on a TTY terminal.

# CONF

Invokes the MUSIC conference facility for discussing topics of interest. For more information refer to the *MUSIC/SP Communications Guide* or type "HELP CONF".

**Syntax:**

```
CONF confname
```

# CONFMAN

Invokes the Conference Manager Utility for setting up and administering a conference. For more information refer to the *MUSIC/SP Communications Guide* or type "HELP CONFMAN".

**Syntax:**

```
CONFMAN
```

# COPY

This command is used to create a copy of a file. The contents and attributes of the original file remain as before.

Use * ("wild" character file name pattern) to specify a group of files. See the examples below.

**Syntax:**

```
COPY oldfile newfile <-REPL> <-SUB>
```

**Parameters:**

oldfile — The name of the file you wish to copy. To copy a group of files, this can be a file name pattern, containing the character "*". The * matches any string of 0 or more characters. The general form is A*B, where A and B are each 0 or more characters. A*B matches any name that starts with A and ends with B.

newfile — The name of the new file. It will be created as an exact copy of the first file, with the same attributes and data contents. If oldfile is a pattern (contains *), then newfile should also be a pattern.

-REPL This optional parameter causes the target file (newfile) to be replaced, without any prompting, if it already exists. Without this option, you are prompted whether to replace the file; reply "y" or "yes" to replace it, "n" or "no" to not do the copy. You can also enter "yes all" (to copy the file and all further files for this command, without more prompting) or "no all" (to not copy the file, or any further files for this command which already exist, without more prompting). To cancel the command, enter "/cancel".

-SUB This optional parameter, when used with a file name containing * (a name pattern), indicates that the matching names in the current (or specified) directory and all lower subdirectories, should be copied. Without this option, the copy operation applies only to files in the current (or specified) directory. The -SUB option should be used when copying an entire directory, or an entire userid. See the examples below.

**Examples:**

`copy work1 work2` Copies the file WORK1 to a new file WORK2. WORK1 still exists after the copy operation.

`copy work* try*` Copies all file names that begin with "work" to the corresponding names beginning with "try". For example, WORK23 is copied to TRY23, WORK.ABC to TRY.ABC, WORK to TRY, etc. Only files in the current directory are copied.

`copy x\abc y\defg` Copies ABC in directory X to a new file DEFG in directory Y. Directory Y must already exist.

`copy ab\* xyz\* -sub`

Copies all files in directory AB to directory XYZ, including all files in lower subdirectories. Target subdirectories are created as needed (including XYZ).

`copy fred:* susan:* -sub`

Copies all files in userid FRED to userid SUSAN. (This form of the command is used only by system administrators.)

# COUNT

The COUNT command displays the usage count and creation date for specified files. Note that counts are maintained only for files that have the count attribute set. The usage count is the number of times the file has been opened since the creation date.

**Syntax:**

```
COUNT  {filename} {pattern} {<listname} {*}
CNT
```

**Parameters:**

filename is the name of the file.

pattern is the file specification including wild characters * and/or ?. Note that a pattern only applies to files in the current directory.

<listname is the name of the file containing a list of file names.

* give the count for all your files that have the CNT attribute.

# DEBUG

This command is used to invoke the DEBUG utility for debugging programs at the machine level. For details see *Chapter 10. Utilities.*

**Syntax:**

```
DEBUG [parameters]
```

*Note:* The DBG command is faster than the DEBUG command and can be used when no parameters are needed.

# DECRYPT

The DECRYPT program is used to restore a file that has been previously "encrypted". ENCRYPT and DECRYPT are two programs that respectively encrypt (code) and decrypt (de-code) files to provide added security. See *Chapter 10. Utilities* for a complete description.

**Syntax:**

```
DECRYPT infile outfile PW(password) REPL(ON|OFF)
```

# /DEFINE

This command defines specified program function (PF) key (on a 3270-type workstation) to represent a string of characters. The subsequent pressing of that key causes the string to be executed or optionally written to the input area to allow for modification of the string before it is executed. It can also be used to redefine the full screen escape key (PA2 by default).

Your function key definitions are active for the current session except when a program using the full screen interface such as the Editor or Mail is running.

**Syntax:**

```
/DEFINE  PFn  [string ]
/DEF     PFn  [=string]
/DEF     Fn   [string]
/DEF     Fn  [=string]
/DEF     ESC  PFn
/DEF     ESC  PAn
/DEF     ESC  TEST
/DEF     ESC  NULL
```

**Parameters:**

PFn
Fn — indicates which Program Function Key to define. *n* can be any number from 1 to 12.

Some models of 3270s have 24 function keys while others have 12. In order to maintain compatibility between models, only PF1 through 12 may be defined. For ease of use, the system automatically maps PF13 through 24 into PF1 through 12.

The function keys are also used to activate functions associated with the Multi-Session and Retrieve support. Specific function keys are allocated for this purpose by default. These default definitions are as follows:

| PF Key | Function | Description |
|---|---|---|
| PF1 | HELP | Access help facility |
| PF4 | ADD | Create a new session |
| PF5 | DELETE | Delete a session |
| PF7 | PREVIOUS | Activate the previous session |
| PF8 | NEXT | Activate the next session |
| PF12 | RETRIEVE | Retrieve the last input entered |

For more information about F4, 5, 7 and 8, see *Chapter 2. Workstations* of this guide.

RETRIEVE (F12)

As commands are entered they are saved in a buffer (except in full screen mode). The RETRIEVE function allows the user to scan back through these commands. Each time F12 (Retrieve) is pressed the previous command is displayed in the input area. The user may enter the command as is, modify it, or continue scanning back by pressing the function key. If there are no more commands in the buffer, the system automatically loops back to display the most recent command.

string — is the *string* of characters to be associated with the specified key. When the key is pressed the *string* is immediately executed.

=string — same as above except the *string* is written to the input area on the 3270 screen. This allows the user to modify or add to the string before pressing the Enter key.

ESC — This keyword is used to redefine the full screen escape key. This is usually PA2 by default and is used during fullscreen applications to access the multi-session function keys. See Note 1. In addition to function and PA keys, the escape key can be defined as TEST or

NULL. TEST indicates that the TEST-REQUEST or SYS-REQUEST key is to be used as the escape. NULL means that no escape key is recognized.

*Notes:*

1. The Multi-Session functions assigned to F4 (ADD), F5 (DELETE), F7 (PREVIOUS), and F8 (NEXT) can be accessed during full screen programs such as the Editor and MAIL. Press the full screen escape key (PA2) before pressing the function key. Any other definitions assigned to function keys do not work during these types of programs.

2. In the full screen applications, especially the Editor, it is important to note that changes to the current screen are not recorded when you press the full screen escape key (PA2). Make sure that you have pressed an action key (i.e. ENTER) before switching sessions.

3. In *Go mode, the program SHOWPFK can be executed to display your current function key definitions.

4. A DEFINE utility program is available for defining a number of function keys at one time. This utility could, for example be set up as an AUTOPROGRAM to assign non-standard default definitions by creating a file, which includes the DEFINE utility followed by the required /DEFINE commands. Whenever that file is executed, the definitions are made. The following is an example of such a file.

```
/INC DEFINE
/DEF PF1 HELP
/DEF PF2 =EDIT
/DEF PF3 OFF
/DEF PF9 MAIL 1
```

5. From within a program, the NXTCMD subroutine can be used to define function keys for the remainder of the session. For example:

```
CALL NXTCMD('/DEFINE PF1 HELP',16)
```

   The second argument is the length of the command string in the first argument.

**Example:**

```
*Go
def pf1 help
*OK
def pf2 =edit
*OK
```

# DELETE

This command is used to delete an extra MUSIC session on a 3270-type workstation. (A session that was previously added with the ADD command.) By default the F5 key has this definition. The DELETE command can only be used in *Go mode for deleting sessions. You cannot use this command if it is the last session or while using a full screen program like the Editor. See *Chapter 2. Workstations* under the heading "Multi-Session Support" for more information.

**Syntax:**

```
DELETE
DEL
```

*Note:* If a file name parameter is added, then this command works like the PURGE command and deletes files. See the PURGE command description for more information.

# DIR

This command is used to obtain the names of the files in your current directory, or some other directory that you specify. The list is produced in alphabetical order. You can use the F or X option to display attributes of the files, in addition to their names. Use the LIBRARY command to list the file names in your entire library, rather than in a single directory.

**Syntax:**

```
DIR searchspec [parameters]
```

See the LIBRARY command for a description of the parameters.

"searchspec" specifies either a directory name, or a file name pattern indicating which names should be listed. A pattern contains 1 or more "wild" characters * and ?.

In the listing, directory names are indicated by "<DIR>".

Additional parameters for the DIR command:

| | |
|---|---|
| SUB | Causes all subdirectories to be included in the search. This lists files in the entire subtree. For example, to find all occurrences of the file ABC in the current directory and all subdirectories, type: dir *abc sub |
| FNAME | Causes full file names (including userid and directory path) to be put out. |
| NODIRN | Causes the names of subdirectories to be omitted from the listing. |
| /W | This option provides compatibility with DOS. It lists the file names in packed form, several per line. It is equivalent to the PACK option. |
| /P | This option provides compatibility with the DOS option for pausing after each screen of output. It is ignored by MUSIC, since screen pausing is automatic. |

**Usage on Batch:**

To run the DIR command in a batch job, use the following control statements:

```
/PARM searchspec <options>
/INC *COM:DIR
```

**Examples:**

| | |
|---|---|
| `dir` | Lists the names of files in your current directory. |
| `dir /w` | Lists the files in packed form, several per line. |
| `dir \` | Lists the files in your root directory. |
| `dir\*.doc` | Lists files in your root directory whose names end in ".DOC". |
| `dir abc* x` | Lists names in your current directory that start with ABC, and requests additional information about each file. |
| `dir test` | Lists the names of files in the TEST directory (which is assumed to be a subdirectory of the current directory) |
| `dir \mydir p` | Lists the names of files in the MYDIR directory, in packed form (several names per line) |
| `dir *\` | Lists the names of all subdirectories in the current directory. |
| `dir * sub` | Lists the names of all files in the current directory and all subdirectories of it. |
| `dir *abc sub` | Lists all occurrences of the file ABC in this directory and all subdirectories. |

# /DISCON

This command disconnects your workstation from the active session while leaving a program running. The program continues to run until one of the following events occur: 1) the program requests input from the workstation; 2) the program's output exhausts the supply of output buffers or; When one of the above events occur, the program stops execution until the workstation is reconnected.

To reconnect to the disconnected session, use a /ID command specifying the original userid. The reconnection must be made from the same type of workstation. For example, if the user was on a 3270-type workstation, then the reconnection must also be made from a 3270-type workstation, not an ASCII one.

**Syntax:**

```
/DISCON
/DISC
```

*Notes:*

1. This command is not valid in command mode.

2. If a userid has been restricted from using the Multi-Session feature then this command is not allowed.

3. Each installation defines a limit to the number of simultaneous users who can use the disconnect and

Multi-Session features. When this limit is exceeded then the /DISCON command is rejected.

# DISPLAY

This command causes all or part of a file to be displayed on the workstation. Each line displayed is preceded by a 4 digit line number. This line number is not stored as part of your file, nor does it have any special significance except to show the order that they are stored in the file. Line numbers beyond 9999 will display only the last 4 digits. Files can be listed without numbers by the use of the LIST command. You cannot use this command from batch, though you can obtain listings with or without line numbers by using the UTIL program described in *Chapter 10. Utilities* of this guide.

**Syntax:**

```
DISPLAY  [filename] [x     ] [y     ]
D                  [LAST  ] [LAST  ]
                   [LAST-n] [LAST-n]
```

**Parameters:**

| | |
|---|---|
| filename | the name of the file to be displayed. If not specified, the Input file is displayed. |
| x | is the line number (or the first line number of a group of line numbers) of the file to be displayed. |
| y | is the last line number of a group of line numbers to be displayed. If *y* is larger than the number of lines in the file, the display continues to the end of the file.<br><br>If *x* or *x* to *y* is not specified, the entire file is displayed starting at the beginning. |
| LAST | is used in place of *x* and/or *y* refers to the last line of the file. LAST-1 refers to the second to last line, etc. |

**Examples:**

```
*Go
display sample
*In progress
0001 LINE 1
0002 LINE 2
0003 LINE 3
*End
*Go

display sample 2 3
*In progress
0002 LINE2
0003 LINE3
*end
*Go
```

*Note:* The parameters for this command may appear in any order.

**Messages:**

`REQUEST OUT OF RANGE`
The file to be displayed is a null file or the specified line number is out of the range in the file.

# DW370

This command is used to invoke the optional IBM DisplayWrite/370 word processing facility. For details see the *MUSIC/SP Mail and Office Applications Guide*.

**Syntax:**

```
DW370 filename [options]
```

# EDIT

The EDIT command is used to invoke the Editor program to edit a file. For a description of various Editor commands, enter HELP or "HELP cmd" (*cmd* is the name of the Editor command) when you are in edit mode of the Editor (not MUSIC command mode). For more information about the Editor and this EDIT command refer *Chapter 7. Using the Editor* of this guide. See also TEDIT and BIGEDIT.

**Syntax:**

```
EDIT [filename] [NEW] [LRECL(n)] [fmt] [RO] [NOLOG] [UIO] [MAX(n)]
E                    [n       ]
```

**Parameters:**

filename — is the name of the file to be edited. If *filename* is omitted, the Input file is assumed and other parameters must be omitted.

NEW — If NEW is specified after *filename* then the Editor assumes the file being edited is a new file and goes to input mode directly. If not specified, OLD is assumed. If *filename* is excluded and NEW is specified, then the Editor goes directly to input mode. (This file will not have a name until you specify one with either the NAME, SAVE, FILE, or EXEC Editor commands.)

LRECL(n) — is the record length to be used during edit. It must be a number from 1 to 8192. If not specified, the record length of the file being edited is used, or 80 for a new file. LRECL(*n*) can be abbreviated to simply *n*, if *n* has two or more digits.

fmt — is the record format of the file to be edited. It can be F (fixed), FC (fixed compressed), V (variable), and VC (variable compressed). If not specified, the original record format of the named file (if the file is old), or FC (if the file is new) is used.

RO — Causes the Editor to go into Read Only (RO) mode. This is equivalent to the BROWSE

command. In full-screen mode, the text on the screen is protected (non-modifiable). Note that RO also implies NOLOG.

NOLOG — Suppresses the Editor restart feature by not looking for a log file or creating a new one.

UIO — Causes the Editor to read the file by 512-byte blocks, using MFIO UIO requests. Each block becomes a record for the edit. This lets you edit (but not change) record format U files, or files that cannot be read sequentially. The record length for the edit is automatically set to 512. UIO implies RO and NOLOG. Do not use the FILE or SAVE Editor commands to write to the file being edited, since UIO is not used for the writes.

MAX(n) — Specifies the maximum number of records the Editor is to read from the file being edited. This option implies RO and NOLOG. It is useful for looking at the first part of a large file. For example: EDIT BIGFILE MAX(500).

**Examples:**

`EDIT MYPROG` edits the file called MYPROG.

`E X NEW 100` edits a new file called X. The record length of the file is 100.

**Messages:**

`*** FILE NOT FOUND`
The file to be edited cannot be found in the Library. Make sure that the file name specified on the EDIT command is correct.

# ENCRYPT

The ENCRYPT program is used to code a file for added security. ENCRYPT and DECRYPT are two programs that respectively encrypt (code) and decrypt (de-code) files. CAUTION: If the encryption password is forgotten, there is no way to restore the file to its original form. See *Chapter 10. Utilities* for a complete description.

**Syntax:**

```
ENCRYPT infile [outfile] [PW(password)] [REPL(ON|OFF)]
```

# ERASE

Same as the PURGE command. See the description for the PURGE command for information.

# EVIEW

Invokes the VIEW program for viewing encrypted files.

**Syntax:**

```
EVIEW [filename] [password]
```

# EXECUTE

This command is used to run a program from a file (execute the file). This file should contain the appropriate job control statements control program execution. See *Chapter 6. MUSIC Job Control Statements* for details. The command scanner defaults to EXECUTE if it does not recognize the input as a valid MUSIC command. So the command verb (EXECUTE), can be omitted from in front of the filename. Unless the filename happens to match a MUSIC command the file can be executed simply by typing its name.

**Syntax:**

```
EXECUTE  filename  [ppppp]
EX

filename  [ppppp]
```

**Parameters:**

filename — The name of a file which contains job control statements to run the program.

ppppp — If specified, it is the parameter string which is to be passed to the program. The use of "EXEC filename ppppp" is logically equivalent to "EXEC SAMPLE" where SAMPLE is the name of a file containing the following statements:

```
/PARM ppppp
/INCLUDE filename
```

A /PARM statement is only generated if the *ppppp* field is specified.

**Example:**

```
*Go
exec hello
*In progress
      (output)
*END
*Go

hello                (same as "exec hello")
*IN PROGRESS
      (output)
```

*Note:* If *filename* does not correspond to any MUSIC command or command abbreviation, the EXEC part can be omitted (unless the user's profile disallows this). This usage is called the implied EXEC command.

**Messages:**

`XXX ERR11 FILE NOT ACCESSIBLE`
The file XXX cannot be found or the user is not allowed to access the file. Make sure that the file name specified on the EXECUTE command is correct.

# FINDTEXT

FINDTEXT is used to search through some or all of the files in your library for a text string. It produces a list of text lines, with line numbers, when the text is found. The file name is also reported.

FINDTEXT supports both full screen and line mode usage. If you want to enter all parameters on screen fields, enter FINDTEXT or FT without parameters. Otherwise a screen is only provide, to assist you in correcting parameters that are in error.

**Syntax:**

```
FINDTEXT 'text'[FILE(spec)] [FROMLINE(n)] [TOLINE(n|END)]
FT             [FROMCOLUMN(n|END)] [TOCOLUMN(n)] [FIRST(YES|NO)]
               [FINDS(n|ALL)] [CASE(I|R)] [OUTPUT(filename)]
```

**Parameters:**

'text' text is a character string that is to be searched for in the list of files defined by FILE. If invoked from *Go, the quotes are required when any of the options below are also specified.

File(spec) *spec* can be one of:
a) a library pattern such as "*.s", "*", or "*work.*.?" etc. All file names in your library that match the specified pattern will be searched for "text".
b) a file that contains a list of files to be used in the search of "text", specified as "<file-name" where filename is an existing file. If this parameter is not specified then all your files are checked.

FROMLine(n|END)
*n* is an integer greater than 0, that specifies the starting line within each file that the search is to begin at. The default is 1. (Abbreviations: FROML and FL.)

TOLine(n|END) *n* is an integer greater than 0, that specifies the last line within each file that the search is to stop at. Keywords "all", "max", and "end" are used to indicate the entire file. The default is end. (Abbreviations: TOL and TL.)

FROMColumn(n) *n* is an integer greater than 0, that specifies the starting column within each line of the file that the search is to begin at. The default is 1. (Abbreviations: FROMC and FC.)

TOColumn(n) *n* is an integer greater than 0, that specifies the last column within each line of the file that the search is to stop at. Keywords "all", "max", and "end" are used to indicate the entire file. The default is end. (Abbreviations: TOC and TC.)

FIRst(YES|NO) When set to "yes", causes the search to stop at the very first match. The default is no.

FINds(n|ALL) specifies the number of times to search within each file or all lines. After *n* matches in a

file, searching is halted in that file. The default is 1.

Case(I|R) When set to I (ignore) the matching is done as if all characters in "text" and the file were in the exact same case. So that "Case" will match with the string "case". When R (respect) is used "Case" will not match "case". The default is ignore.

Output(filename) This option defines where the output of the search will be placed. You can enter here any file name. The default is "*terminal" to display output at your workstation.

**Examples:**

1. In this example the string "call tstime(" will be searched for in the library files that end in ".s" .

```
FT 'call tstime(' f(*.s)
```

2. This example searches for the string 'montreal' in the files that end in ".doc" and stores the output of the search in file LIST. Since we want to find only lower case "montreal", we will set case to respect.

```
ft 'montreal' f(*.doc) c(r) o(list)
```

3. In this example we will locate and display only those files where "/inc gork" occurs on line 5 of the file.

```
ft '/inc gork' froml(5) tol(5)
```

# 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.

Note that you can create a "plan" file, using the PLAN command. When remote users query your userid on MUSIC, the contents of this file is transmitted to them.

**Syntax:**

```
FINGER [/W ][user-name]@site-name[@site-name...]
```

**Parameters:**

user-name is the name of the login ID that you are querying. If this is omitted, then information about all users logged in is displayed. Note that without a '@' present, a local user-name is assumed.

/W is an option sent to the remote host requesting more detail in the displayed information. (Note: the remote host may ignore this request.) A blank **must** follow this option.

@site-name is the name of the remote site to send the query to. Note that you may have a number of sitenames strung together. In this case, FINGER sends all of the text to the right of the last

'@' character to the remote host - it will do the same - and will then display the information that it receives from the remote host. Note that you should fully qualify all of leftmost addresses, as other systems may not extend the address with the local site's domain name as MUSIC does. (Note that site-name may be set to '*' - this indicates the local site (only valid for the right-most address.)

*Note:* Note that the @ character must be specified in order to indicate an address; otherwise, the local site is assumed

**Examples:**

```
all users at local site:    finger *

user xx00 at local site:    finger xx00

user fred at remote site    finger fred@remote.site.edu

user fred at remote site,
extended info requested:    finger /W fred @remote.site1.edu

user fred at site 1 via     finger /W fred @rmt.site1.edu@rmt.site2.edu
site2:                      finger /W fred @rmt.site1.edu@rmt.site2.edu
```

# FLIB

This command lists the names for your files and directories, and allows selection of these names for browsing, editing, renaming, copying, etc.

**Syntax:**

```
FLIB filespec
```

You can invoke this program by using the FLIB command, selecting the file management item on the FSI main menu, or invoking FSI with a file specification. Include the *filespec* parameter to specify which group of files you wish to access; otherwise the same *filespec* from the last FLIB or FSI command is used again. Help is provided once this program is invoked.

# FSI

This command is used to invoke the FSI (Full Screen Interface) subsystem. This facility allows you access to various components of MUSIC system through a series of selection menus. For details see *Chapter 1. Introduction*.

**Syntax:**

```
FSI [filespec|n]
```

# FTP

This command is used to invoke the File Transfer Protocol program for transferring files between sites that are connected via TCP/IP to the Internet. A list of Internet addresses that allow anonymous access is supplied. A brief description of FTP can be found in *Chapter 10 - Utilities* under "FTP and TELNET from MUSIC". For full details refer to the *MUSIC/SP Communications Guide*.

**Syntax:**

```
FTP [internet address]
```

Help is provided once this program is invoked.

# GDDM

This command is used to invoke the optional IBM GDDM graphics utilities. For details see *Chapter 8. Processors*.

**Syntax:**

```
GDDM
```

# GETMAIL

This command invokes the GETMAIL utility program to read your incoming mail and store the text in a file.

**Syntax:**

```
GETMAIL filename options
```

| | |
|---|---|
| filename | is the name of the file for storing the text of all of the incoming mail items. A file name must be specified with this command. |
| options | A number of options can be used with this command and are specified after the file name separated by blanks. If an option is repeated the last option specified takes precedence. |

**Options:**

| | |
|---|---|
| APPEND | store the mail text at the end of the file. |
| REPLACE | store the mail text in a new file. |
| DELETE | delete the mail item after it has been processed. |
| KEEP | do not delete the mail item after it has been processed. |

| | |
|---|---|
| N | N is an integer number which represents the incoming mail item to get. (if ALL is used before N, then N is used.) |
| ALL | process all incoming mail items in the mailbox. (if N is used before ALL, then ALL is used.) |
| MSGSON | display error messages. |
| MSGSOFF | suppress the display of error messages. |
| DISCARD | discard the list of information that is supposed to be written to the file. |
| SHRINK | shrink the mailbox to its absolute minimum file size. |
| SELECT | select incoming mail items in the mailbox via a criterion. |
| SELECTN | select incoming mail items in the mailbox via a criterion. |
| FOR(name) | allows you to list the mail information for *name's* mailbox provided you are allowed as a surrogate for *name's* mailbox. This parameter is similar to the MAIL program command FOR. |
| UIDL(n) | get the incoming mail item designated by n, where n represents the unique-id listing, UIDL, for the mail item. (if UIDL(n) is given, it always takes precedence over N and ALL.) |

For more information type "/help getmail"

# GETMINFO

This command invokes the GETMINFO utility program. It gets a list of information about your incoming mail items and stores this in a file.

**Syntax:**

```
GETMINFO filename options
```

**Parameters:**

| | |
|---|---|
| filename | is the name of the file for storing the list of incoming mail. A file name must be specified with this command. |
| options | A number of options can be used with this command and are specified after the file name separated by blanks. If an option is repeated the last option specified takes precedence. |

**Options:**

| | |
|---|---|
| APPEND | store the mail information at the end of the file. |
| REPLACE | store the mail information in a new file. |
| DELETE | delete the mail item after it has been processed. |
| KEEP | do not delete the mail item after it has been processed. |
| N | N is an integer number which represents the incoming mail item to get. (if ALL is used before N, then N is used.) |
| ALL | process all incoming mail items in the mailbox. (if N is used before ALL, then ALL is used.) |
| MSGSON | display error messages. |
| MSGSOFF | suppress the display of error messages. |
| DISCARD | discard the list of information that is supposed to be written to the file. |
| SHRINK | shrink the mailbox to its absolute minimum file size. |
| SELECT | select incoming mail items in the mailbox via a criterion. |
| SELECTN | select incoming mail items in the mailbox via a criterion. |

FOR(name) allows you to list the mail information for *name's* mailbox provided you are allowed as a surrogate for *name's* mailbox. This parameter is similar to the MAIL program command FOR.

UIDLS displays a unique-id listing, UIDL, for each of the selected incoming mail items. Since the UIDL associated with a specific mail item is never reused for the life of the item, using the UIDL is a more exact method to specify a particular item. GETMAIL with the UIDL(n) parameter can be used to get individual mail items.

Here are some examples of GETMINFO commands:

```
GETMINFO NEWINFO APPEND ALL SELECT TYPE NEW

GETMINFO NEWINFO APPEND 1 KEEP

/INC *COM:GETMINFO
NEWINFO APPEND ALL KEEP -
SELECT SUBJECT THIS IS A VERY LONG SUBJECT THAT GOES ON AN ON

/INC *COM:GETMINFO
NEWINFO APPEND ALL KEEP -
SELECT (SUBJECT THIS IS A VERY LONG)&(SEND 11JUL91)

GETMINFO NEWINFO ALL DELETE DISCARD SELECT BEFORE 01APR93

GETMINFO NEWINFO UIDLS KEEP
```

For more information type "/help getminfo"

# GOPHER

This command invokes the Gopher server (provided your site has TCP/IP connections). Gopher is a document search and retrieval system running on the Internet.

**Syntax:**

```
GOPHER [site_address]
```

Help is provided once the program is invoked. For more information, refer to the *MUSIC/SP Communications Guide*.

# HELP

This command invokes the MUSIC HELP facility for accessing information about a wide variety of topics. Most of this guide can be found online through this facility. For example, you can enquire about how to use a particular MUSIC command or a utility program. If the information about a particular item is not available, the item will be recorded in a system log file which will be reviewed by the MUSIC administrator.

Workstations with full-screen display enables you to easily browse the HELP facility through menus and text

screens. You are able to page forward and backward. You can place your cursor on any highlighted topic names to jump from one topic to another.

The HELP command can be used when you are in command mode. If just HELP is entered without a topic, general information about MUSIC is given and a list of general topics is displayed. You can then go into more detail on any of the topics available.

**Syntax:**

```
HELP [topicname n]
```

topicname — is the name of the item on which you want information. *topicname* can be the name of an individual item or the name of a menu of items.

n — *n* is the item selection code (usually a number) from a help menu. *(n* is only used if you know the item selection code in advance.)

**Examples:**

`HELP HOURS` gets the hours of operation of MUSIC.

`HELP TOPICS` displays a list of the available topics under this facility.

*Notes:*

1. The HELP command used from command mode accesses MUSIC's general help facility. Other help facilities are provided with a variety of programs on MUSIC. For example, when you are using FSI (Full Screen Interface), the command "HELP" (or F1) gives you the FSI help facility. If you wanted MUSIC's general help facility while you are within FSI, use the command "/HELP". The slash is necessary to distinguish MUSIC commands from FSI commands.

2. Systems administrators should refer to the *MUSIC/SP Administrator's Reference* for information about updating existing help facilities and creating new ones.

# /ID

This command is used to identify and validate a user signing on the system. It performs the following functions:

- identifies you to the system (by userid).
- asks for the password assigned to your userid. (The area where the password is typed is blacked out by the system.)
- displays system information messages.
- sets up default tabs, etc., if any are defined in your *profile*. (See the PROFILE utility for more information.)
- schedules your AUTO/ALWAYS program (if any exists).

A job control statement form of the /ID command is used when submitting batch jobs. See *Chapter 3. Using Batch* of this guide.

If your userid has a fund restriction, then the funds remaining for the userid as of the last accounting are displayed at sign-on time. When the funds remaining is less than 10% of the total allocation, or $10, whichever is less, a warning message is issued. If the allocation of funds for MUSIC usage have been used up, the message `*USERID OUT OF FUNDS` appears and the user must contact the MUSIC System Administrator for additional funds.

Messages of general interest may be received while signing on. These messages are sent to all workstations and may contain news items, etc, or may instruct you to type the NEWS command to receive more information.

MUSIC automatically requests a /ID command be entered when a workstation first connects to the system. The /ID command can also be entered from command (*Go) mode, should the user wish to change the current userid in mid-session.

**Syntax:**

```
/ID   [tn,]userid[,ident][;trmcls]
```

**Parameters:**

tn — is the terminal (workstation) identification number. This optional parameter may be used to distinguish one session from another for accounting purposes. If specified, this parameter must be a decimal number between 0 and 99 and must be separated from the /ID by at least one blank.

userid — is your 1-16 character userid assigned to the MUSIC user by the MUSIC System Administrator.

ident — is an identification field of up to eight characters. This field is optional unless your userid has been set up to require it.

trmcls — is the terminal (workstation) class parameter which provides MUSIC with additional information about the physical characteristics of your workstation. Rather than always entering this parameter on the /ID command, a default terminal (workstation) class can be set in the user profile. (See TERM parameter of the PROFIL utility). The following lists the workstation classes that are available on the base MUSIC system. Your installation may define other workstation class names that you could use in this field. Those flagged with the * are the defaults for the particular type of workstation and need not be specified on the /ID command.

| trmcls | Terminal Model |
|---|---|
| 3270 | *Any of the 3270 family of terminals |
| 3270A | 3277 with DAF/APL |
| 3270B | 3270 APL/TEXT feature |
| ASCII | *Any ASCII printer or ASCII video display terminal |
| 3101 | 3101 terminal or PC running 3101 emulator |
| PCWS | PC running MUSIC's PC Workstation Software |
| IBMPC | PC running Async Communications Software |
| 2741 | *Any of the 2741 family of terminals |

*Note:* For users with 3270-type workstations, it is important for the Editor to know the actual number of function keys on the workstation being used. On the sign-on screen, if the /ID command and password are entered by pressing the ENTER key, MUSIC assumes 24 function keys for IBM 3178, 3278 or 3279 terminals. Otherwise, MUSIC assumes 12 function keys (e.g., IBM 3277). You can override these assumptions by pressing the highest numbered function key instead of the ENTER key when signing on. (To receive the sign-on screen when you are in *Go mode type /ID.) Refer to the "Editor Full Screen Mode" in *Chapter 7 - Using the Editor* of this guide.

**Examples:**

Example 1 - Sign-on Screen on a 3270-type workstation

```
*MUSIC/SP -- PLEASE SIGN ON

ID Command: /ID _              <-- The cursor is positioned
                                   on this line for typing
  Password:                        your userid.  Use the NEW
                                   LINE or TAB key to skip to
                                   the password field, type
                                   your password, press ENTER
                                   (or F12 or F24).
F1/13 - HELP    F3/15 - /OFF
------------------------------------------------------------

*Userid last signed on 17:10 1993/05/14
*Sign-on 1993/05/14, Time=09:10, Port=08E, TCB=104
*Funds Remaining as of Last Accounting..$168.17
*Go
```

Example 2 - Signing on an ASCII terminal

```
*MUSIC/SP  --  SIGN ON
/id user
*Password?
XXXXXXXX
*Userid last signed on 08:26 1993/01/14
*Sign-on 1993/05/14, Time=10:45, Port=0B0, TCB=019
*Funds Remaining as of Last Accounting..$ 184.52
*Go
```

# IDP

This command invokes the Information Display Program (IDP). This program is used to create help facilities and bulletin boards.

**Syntax:**

```
IDP
```

Help is provided once the program is invoked. For more information, refer to the *MUSIC/SP Campus-Wide*

*Information Systems* (*CWIS*) *Guide*.

# IRC

Initiates the Internet Relay Chat program for communicating with others connected to the Internet.

**Syntax:**

```
IRC
```

Help is provided once the program is invoked. For more information, refer to the *MUSIC/SP Communications Guide*.

# LANG

The LANG command allows you to display or change the default language setting for messages, etc. 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, Spanish. Enter "LANG ?" to get a list of the languages supported at your site.

**Syntax:**

```
LANGUAGE [language] [?]
LANG
```

**Parameters:**

language specifies the language of your choice.

? lists the languages available at your site.

# LIBRARY

This command is used to obtain a list of file names saved in the under your userid. The list is produced in alphabetical order. The abbreviation of each parameter is shown under its full form. Use the DIR command to list the file names in your current directory.

**Syntax:**

```
LIBRARY searchspec [FULL][TAG][VSAM][PACK][NOSORT][SAVE(name)]
LIB                [F   ][T  ][V   ][P   ]        [S(name)   ]

                   [COM][X][SPACE(n)][FNAME][SORT(type)][APPEND]
                                           [SO(type)  ][AP    ]
```

The LIBRARY command can be entered without specifying *searchspec* or parameters. If you wish to add parameters to this command then you must specify *searchspec*. Parameters after *searchspec* can be in any order. They must be separated by 1 or more blanks.

**Parameters:**

searchspec — (search specification) specifies which file names, belonging to the user, are to be searched for in the Save Library index. It may be an actual file name, in which case only that file is listed. Or, the string may contain one or more wild characters ? and *, in which case all file names matching the pattern are listed. A ? matches any single character in the corresponding position of a file name. A * matches any group of 0 or more characters. If the *searchspec* parameter is not specified, all the file names belonging to the user are listed.

To list all your filenames you can enter LIBRARY or if you wish to specify parameters then enter "LIBRARY * parameters". The * is your search specification indicating all files on your userid.

FULL — indicates that for each file listed, its corresponding attributes are also given. See the discussion below on file attributes about the information provided.

X — This option is similar to FULL, but the output is in a slightly different format and includes time of last open for write, userid of last writer, and number of records. An asterisk (*) appears after the file size if the file has releasable unused space.

TAG — is the same as specifying FULL except that it also displays the tag information for each file. TAG implies FULL.

VSAM — lists only VSAM (Virtual Storage Access Method) files. VSAM implies FULL.

COM — lists only files in the common index.

PACK — Normally only one file name is displayed per line for the LIBRARY command. Specifying PACK indicates that several file names may be combined on one line. PACK cannot be specified if a FULL, TAG or VSAM parameter is used. Alternate forms: P, WIDE, W, /W

NOSORT — causes the file names to be put out in unsorted order. This causes output to appear immediately. When the library listing is sorted then there is a short delay.

SORT(type) — specifies how the listing is to be sorted. Type is NAME (sort by file name), SIZE (sort by file size in K), RDATE (sort by date last read), WDATE (sort by date last written), or UDATE (sort by reference date, which is the higher of the read and write dates). Abbreviations are N, S, R, W, U. The default is SORT(NAME). A minus sign (-) can be placed before the type to sort in descending order. If the SORT option specifies a type other than NAME, and neither X nor FULL is used, then the X option is automatically used. Examples: SORT(WDATE), SORT(-S).

SAVE(name) indicates that the output of the LIBRARY command is to be saved in a file instead of displayed on the workstation (unit 6). If *(name)* is specified with the SAVE parameter, the output is written to the file called *name*. If *name* is omitted, the name @LIB is used. The original contents in the file is overwritten if the file already exists. See also the SPACE(n) and APPEND parameters below.

SPACE(n) specifies the initial space (in K) to be allocated for the new file specified by the SAVE parameter. The default is SPACE(32), meaning 32K.

APPEND specifies that the output should be written to the end of the file given by the SAVE parameter, after any existing data. This is useful for accumulating the output of several LIB or DIR commands into a single file. If the output file does not already exist, a new one is created. In all cases when the APPEND option is specified, unused space in the output file is NOT freed at the end of the command; this is in anticipation of further appends to the file. Without APPEND, unused space is freed.

FNAME Causes full file names (including userid and directory path) to be displayed.

**File Attributes**

When FULL is specified on the LIBRARY command, extra information about each listed file is also displayed (See examples below). This information consists of:

name indicates the name of the file.

Rsiz indicates the logical record length of the file.

Rfm indicates the record format of the file. The possible record formats are F (fixed length), FC (fixed compressed), V (variable length), VC (variable compressed), and U (undefined).

Size indicates the size of the file in number of K (1024) bytes incremented in 2K bytes. The smallest size for a file 2K.

Used indicates the percentage of the file space that is used.

Ext indicates the number of extents of disk space that are used by the file.

Ref indicates the date the file was last opened for reading only. 0000000 means that the file has not been referenced since it was last written on.

Write indicates the date the file was last opened for writing.

T indicates the type of file. The file is in the common library (public) when the letter C appears in this column. If a V appears, the file is a VSAM file.

Own indicates the access control of the file for the owner. R means read access is allowed. W means write access is allowed. X means only read access for execute-only is allowed. A means only write access for append is allowed.

Other indicates the access control of the file for non-owners. The meaning of various letters is the same as listed above.

**Usage on Batch:**

To run the LIBRARY command in a batch job, use the following control statements:

```
/PARM searchspec <options>
/INC *COM:LIB
```

**Examples:**

```
*Go
library
*In progress

Files - CCXA 21OCT91    11 Files
   1 @ELOG.000
   2 ALICE
   3 BOB
   4 CAROL
   5 DATAXX
   6 FIL18
   7 KETTLE.S
   8 MISC\
   9 MISC\FILE1
  10 MISC\FILE2
  11 ROY
*End
*Go

lib * s(mylib)
*In progress

11 filenames saved to file MYLIB
*End
*Go

lib * full
*In progress

Files - CCXA 21OCT91    12 files
    File Name        Rsiz Rfm  Size Used Ext  Ref    Write T  Own Other
1 @ELOG.000            72 VC     2K   0%  1 21OCT91 21OCT91   RW
2 ALICE                80 FC     2K  33%  1 15OCT91 12OCT91 C R
 etc.
Total size of the listed files is  68K
```

*Note:* Files that begin with @ are often generated and regenerated by the programs that you use. For example, @ELOG.000 is the Edit log file created by the Editor program.

# LIST

This command displays all or part of the contents of a file. This command is similar to DISPLAY except no line numbers are shown. This command cannot be used in a job run from batch. Instead you may use the UTIL program from batch to accomplish a similar function. The UTIL program is documented in *Chapter 10. Utilities* of this guide.

**Syntax:**

```
LIST  [filename] [x     ] [y     ]
L               [LAST  ] [LAST  ]
                [LAST-n] [LAST-n]
```

**Parameters:**

| | |
|---|---|
| filename | the name of the file to be listed. If not specified, the Input file is listed. |
| x | is the line number (or the first line number of a group of lines) of the file to be listed. |
| y | is the last line number of a group of line numbers to be listed. If *y* is larger than the number of lines in the file, the list continues to the end of the file.<br><br>If *x* or *x* to *y* is not specified, the entire file is listed starting at the beginning. |
| LAST | is used in place of *x* and/or *y* and refers to the last line of the file. LAST-1 refers to the second to last line, etc. |

**Example:**

```
*Go
list sample
*In progress
LINE 1
LINE 2
LINE 3
*End
*Go
list sample last
*In Progress
LINE 3
*END
*Go
```

*Note:* The parameters for this command may appear in any order.

**Messages:**

`REQUEST OUT OF RANGE`
The file which is to be listed is a null file or the specified line number is out of the range of the file.

# LM

This command invokes the List Manager facility for managing your subscriptions to discussion lists.

**Syntax:**

```
LM
```

Help is provided once this program is invoked.

# MAIL

This command invokes the Electronic Mail Facility for sending and receiving mail. Help is available once the facility is invoked. For complete information see the *MUSIC/SP Mail and Office Applications Guide*.

**Syntax:**

```
MAIL [n]
```

# MAKxxxx

The MAKxxxx commands allows you to change the attribute of a file.

**Syntax:**

```
MAKAPPO filename
MAKCOM filename
MAKPRIV filename
MAKPUBL filename
MAKSHR filename
```

*Note:* Any of the MAKxxxx commands can be typed in the margin area on the FSI file management screen, to change the attributes of the corresponding file.

**Example:**

```
makshr myfile
```

See the CHMOD command for details about file attributes.

# MAN

This command invokes the word search facility for displaying MUSIC manuals. Help is provided once the program is invoked.

**Syntax:**

```
MAN
```

# MD

This command makes a new directory.

**Syntax:**

```
MD dirname
```

The *dirname* parameter specifies the file name for the directory. Directories contain names of files that you want to keep together as a group. The CD command is used to specify the name of your current directory.

Notice that you must issue the CD command to actually use the directory that you create.

Directories can be established within directories. For example, you can have a directory called PARTS within a directory called CAR. You can use the command "MD PARTS" to make a directory called PARTS under your current directory.

Use the RD command to remove directories that you have created.

Use the TREE command to visually show the names of your directories and how they relate to each other.

**Examples:**

```
*Go
md car

*Go
cd car

*Go \CAR>
md parts

*Go \CAR>
cd parts

*Go \CAR\PARTS>
```

# MEET

Invokes the TODO program called MEET. It works in conjunction with the SCHEDULE program to help schedule a meeting by: finding free time in the schedules of available rooms and attendees; adding the meeting to these schedules; and sending mail. See the *MUSIC/SP Mail and Office Applications Guide* for details.

**Syntax:**

```
MEET        or        TODO M
```

## MESSAGES

This command controls whether you are receiving messages or not. By default messages from other users and programs such a MAIL are displayed in a pop up window as soon as they arrive. These can be suppressed by specifying the OFF option. Messages from the system operator cannot be suppressed.

**Syntax:**

```
MESSAGES [ON]
M        [OFF]
```

## MNSORT

This command is used to invoke the MNSORT utility program for sorting data on disk or tape. (See the SORT command for sorting data in a file.) For details about MNSORT see *Chapter 10. Utilities* under the topic "Sorting".

**Syntax:**

```
MNSORT
```

## MS

This command is used to display information about the MUSIC sessions that you are currently signed onto. The terminal (workstation) ID, your userid, and the last command entered for each session are listed.

**Syntax:**

```
MS
```

# NET

This command displays a list of network nodes and allows you to make a connection using the FTP, GOPHER, or TELNET commands. For more details see *Chapter 10 - Utilities* under "FTP and TELNET from MUSIC". For complete documentation see the *MUSIC/SP Communications Guide*.

**Syntax:**

```
NET
```

# NEWPW

This command is used to change your MUSIC sign-on password.

**Syntax:**

```
NEWPW
```

# NEWS

This command is used to list current news items. This news facility is used to communicate items of interest to MUSIC users such as hours of operations, new programs available, etc. Some installations may choose not to maintain this list.

The items in this news file are arranged so that the newest item is displayed first.

**Syntax:**

```
NEWS
NEW
```

**Example:**

```
*Go
news
*In progress

CURRENT NEWS - MAY 14, 1975     12.25.05

......
......
```

## /NEXT

This command is used to go to the next session in the extra session chain. (Extra sessions are added with the ADD command.) By default the F8 key has this definition in command mode. To access the next session while using a full screen program like the Editor, press the full screen escape (PA2) before the F8 key. See *Chapter 2. Workstations* under the heading "Multi-Session Support" for more information.

**Syntax:**

```
/NEXT
```

## OFF

This command is used to terminate a workstation session and to close off accounting for that session. Once the OFF command is entered the system disconnects the hook up (ie. telephone line) to the computer. The connect time and service units used are displayed giving a rough indication of the cost for the session. The charge for I/O to the workstation and the surcharge for larger user regions are not included in the number of service units used.

**Syntax:**

```
OFF  [HOLD]
```

**Parameters:**

HOLD — If OFF HOLD is specified, the workstation session is terminated and workstation specifications such as input/output tab settings, tab and backspace characters, are all reset. The workstation remains connected to MUSIC for a short period of time so a new /ID command can be entered to begin another workstation session. In other words, you do not have to dial MUSIC before signing on.

**Example:**

```
*Go
off
*Good-bye. Connect =  01:25, S.U. =    24
```

## OUTPUT

This command is used to invoke the OUTPUT program to inspect output sent back to MUSIC from batch jobs. A full description of the OUTPUT program is given in *Chapter 3. Using Batch* of this guide. Online help is available once the program has been invoked.

**Syntax:**

```
OUTPUT
```

# PCEXEC

Executes the specified DOS command or PC program as though it had been entered at the DOS prompt. For complete details, refer to the *MUSIC/SP Personal Computer WorkStation User's Guide*, or type "HELP PCEXEC".

**Syntax:**

```
PCEXEC command [-Hold] [-Direct]
               [-H   ] [-D     ]
```

# PHONE

Invokes the TODO program called PHONE. It keeps a log of your telephone conversations. See the *MUSIC/SP Mail and Office Applications Guide* for details.

**Syntax:**

```
PHONE        or         TODO 3
```

# PING

The PING (Packet Internet Groper) command measures round-trip-times to Internet sites. For a brief explanation about TCP/IP and Internet see *Chapter 10 - Utilities* under "FTP and TELNET from MUSIC". For full details refer to the *MUSIC/SP Communications Guide*.

**Syntax:**

```
   PING site-name #packets <-V -D>
```

**Parameters:**

-v (verbose) describes, in more detail, the operation of the program.

-d (dump) provides -v output as well as dumping extra information.

# PIPE

This command provides access to pipe-lines.

**Syntax:**

```
PIPE xxxxxxxx
```

See the section in *Chapter 8 - Processors* or type "HELP" for more information.

# PLAN

Invokes the Editor for updating your @PLAN file. This file is used if someone sends the FINGER command to your site about your userid.

**Syntax:**

```
PLAN
```

For more information, refer to the *MUSIC/SP Communications Guide* or type "HELP FINGER" in *Go mode.

# POLYSOLVE

This command invokes the MUSIC/SP POLYSOLVE program for solving calculations and equations. For more information see *Chapter 10. Utilities*.

**Syntax:**

```
POLYSOLVE
POLYSO
```

# POST

The POST command allows you to submit an article to one or more newsgroups. Fill in the appropriate fields on the screen. Help is provided once the POST command is used.

**Syntax:**

```
POST
```

# PQ

This command is used to find out what is queued to print on various printers.

**Syntax:**

```
PQ
```

# /PREVIOUS

This command is used to go to the previous MUSIC session on a 3270-type workstation. (Extra sessions are added with the ADD command.) By default the F7 key has this definition in command mode. To access the previous session while using a full screen program like the Editor, press the fullscreen escape (PA2) before the F7 key. See *Chapter 2. Workstations* under the heading "Multi-Session Support" for more information.

**Syntax:**

```
/PREVIOUS
/PREV
```

# PRINT

This command schedules the printing of a file to a specified printer. (More information about PRINT is given in *Chapter 3 - Using Batch*.) The file name must be the first parameter. The other parameters are optional and may appear in any order. Carriage control is added to skip to a new page every sixty lines unless CC is specified or the file's record length is 121 or 133.

The PRINT command does not send your file for execution. It prints the contents of the file. The Editor also has a similar PRINT command.

**Syntax:**

```
PRINT filename [ROUTE(printername)] [FORMS(x)] [COPIES(n)] [CC  ]
                R                    F          C          [NOCC]

               [PAGELEN(m)]
                P
```

**Parameters:**

filename — The name of the file to be printed. Under the Editor, the special name **CUR* indicates the current contents of the file being edited, and the special name . indicates marked lines.

printername — The name of the printer where the file is to be printed. This may be an actual printer name or a printer location. It is 1 to 8 characters long. Some documentation refers to this as a "route name" or "routing name". The names are assigned by your system administrator.

If you do not specify a printer name, a default name is used. If you have used the command "ROUTE printername" previously in this MUSIC session, that name is the default. Otherwise, the name defined by ROUTE(name) in your User Profile (the PROFILE command) is used, if any. Otherwise, a default name based on your workstation location may be used If none of the above cases apply, the name SYSTEM is used.

The following names are valid:

SYSTEM — Sends the output to the standard system printer.

MUSIC — Sends the output to the MUSIC Output Queue (the OUTPUT Facility).

DUMMY — Discards the output. Nothing is printed.

rscsname — The name of an RSCS printer. For example: R(PRINTER3) means the output is sent to RSCS and queued for printing on linkid PRINTER3.

prtname — The name of a MUSIC-controlled ASCII or 3270 printer as defined by your installation. Consult your installation for a list of valid names.

PC1 — A printer name such as PC1 may be defined by your installation. It prints the file on your PC printer (using DOS device LPT1), provided your PC is connected to MUSIC via NET3270 or PCWS. If the connection does not support PC printing, the data is sent to the MUSIC Output Queue (the OUTPUT Facility). Similarly, PC2 uses device LPT2 and PC3 uses device LPT3. When printing to a PC printer, some PRINT parameters such as COPIES, FORMS and PAGELEN may be ignored.

n — The number of copies that should be printed. The default is 1 copy.

x — A one to 8 character string indicating which forms are to be used when printing the file.

m — The number of lines per page. The default is 60 lines. This parameter is used only when the NOCC parameter is in effect.

CC — If specified, it indicates that the file to be printed already contains a carriage control charac-

ter on the first character of each line in the file. The PRINT command attempts to honour these control characters.

NOCC Indicates that the file does not contain printer control characters. The file is printed single spaced, with a skip to a new page after each m lines of output. NOCC is the default, except when the record length of the file is 121 or 133, in which case CC is assumed.

**Example:**

```
*Go
print file1
*In Progress
15 records scheduled to print, route SYSTEM
*End
*Go
```

# PROFILE

This command invokes the User Profile Program for changing such things as your sign-on password, default tab characters, job time limits, etc. For complete information see *Chapter 10. Utilities*.

**Syntax:**

```
PROFILE
PROFIL
```

# PROG

This command is used to invoke the PROG (Programmer's Menu) subsystem. This facility allows you access to various components of MUSIC system through a series of selection menus. For details see *Chapter 1. Introduction*.

**Syntax:**

```
PROG
```

# PURGE

This command permanently removes (deletes) one or more files. Files are protected by the first 4 characters of your userid. (You can only purge files that you own; files created by your userid.) The PURGE command does not work with files owned by another userid. The ERASE and DELETE commands (with a file name as a parameter) do the same as PURGE.

The parameters (1 or more) indicate which files to delete and any special options. The Editor also has a

similar command.

**Syntax:**

```
PURGE  filespec <filespec>... <-NOPROMPT><-NOLIST><-SUB>
PUR                           <-NOP     ><-NOL   ><    >
```

**Parameters:**

filespec Each filespec can be one of the following 3 types:

1. The name of a file. The special names /INPUT, /REC and /HOLD may also be used.

2. A file name preceded by the character <, for example, "PURGE <fileabc". Where fileabc is the name of a file that contains a list of file names to be deleted. Each name must start in column 1 and be followed by at least blank. Remaining characters in the line are ignored. Note that the LIBRARY command, with the SAVE(filename) option, can be used to create this list file.

3. A file name pattern (also called a generic file name). The purge command searches the current (or specified) directory for all file names which match the specified pattern and purges them. A pattern is similar to a normal file name, except that it contains one or more of the special wild characters ? and *. A ? in the pattern is considered to match any single character in the corresponding position of a file name. A * in the pattern matches any group of 0 or more characters in a file name. Some examples are shown below.

   Before the files are purged, the matching file names are displayed and you are prompted for permission to proceed. You may respond by typing PURGE (or PUR or PU) to allow the purges, or HELP. Any other response, such as a blank line or /CANCEL, cancels the request and the files are not deleted. The names are displayed in alphabetical order.

-NOPROMPT (abbreviation -NOP or -NOPR) Causes files to be purged immediately, without prompting, when a pattern is specified. Note: for a PURGE command in a batch job, -NOPROMPT is always assumed.

-NOLIST (abbreviation -NOL) Suppresses the message which is normally given for each file purged. However, a message is always displayed if a file cannot be purged (e.g. because it does not exist or is in use).

-SUB This option, used with a filespec that is a pattern, causes the search to include all sub-directories below the current (or specified) directory, in addition to the current (or specified) directory. It can be used to delete an entire subtree. Without this option, deletes are limited to the current (or specified) directory.

**Examples:**

| | |
|---|---|
| **purge myprog.obj** | Deletes the single file MYPROG.OBJ. |
| **pur fila filb filc** | Purges 3 files. |
| **purge <mylist** | Purges the files whose names are in the file MYLIST. |
| **purge abc??** | Purges all files whose names are 5 characters long and start with ABC. |

| | |
|---|---|
| **purge prog1.* prog1** | Purges all files whose names start with PROG1. and also the file PROG1 itself. |
| **purge *old*** | Purges all files that contain the string OLD anywhere in the name. |

# QFTP

This command invokes the Quick FTP menu for transferring files. It is a fast method of using FTP when you know the names of the files you want to get.

**Syntax:**

```
QFTP
```

# RD

This command removes directories.

**Syntax:**

```
RD dirname
```

The *dirname* parameter indicates which directory to remove.

Directories contain names of files that you want to keep together as a group. The MD command is used to make directories.

Notice that you cannot remove a directory that has any files in it. You can use the command DIR to show you the names of the files in the directory.

You cannot remove the name of a directory that is part of your current directory. For example if you are in the directory "\CAR\PARTS" then you cannot remove either the CAR or the PARTS directory.

Use the TREE command to visually show the names of your directories and how they relate to each other.

**Examples:**

```
*Go
md car

*Go
cd car

*Go \CAR>
cd \
```

```
*Go
rd car

*Go
```

# /RECORD

This command is used to control writing to the session recording file. The name of this file is @REC (or @REC.sub where *sub* is the subcode for your userid). It can also be accessed via the name "/REC". When recording is set on, all workstation input and output associated with application programs is added to the file except for I/O associated with full screen programs such as the Editor. System messages resulting from STATUS commands and messages from other users or the system operators console are not recorded.

Recording can be turned on and off as required. Subsequent recorded information is appended to the end of the /REC file unless the NEW option is specified.

Each output record on the recording file starts with a carriage control character. Input records begin with a greater than sign > to distinguish them from the output records.

The recording file is subject to the user's file space limits. Exceeding these file limits causes the system to issue a message and turn off the recording without affecting the execution of the program.

**Syntax:**

```
RECORD  [ON ]
REC     [OFF]
        [NEW]
```

**Parameters:**

ON Turn recording on. If there is text in the file /REC from a previous recording session, then new information is appended to this file.

OFF Turn off recording (stop writing to the /REC file).

NEW Turn recording on. Any text from a previous recording session is deleted and a new /REC file is created.

**Messages:**

`RECORDING TERMINATED.`
This message appears if you run out of space.

*Note:* You can use the PRINT command with the CC option to print the recording file. For example,

```
PRINT /REC CC
```

# RENAME

This command causes a file to be renamed to a specified new name.

**Syntax:**

```
RENAME   oldname newname [-REPL] [-SUB]
REN
```

You may only change the names of the files which you own (files created by your userid). Use * to specify file groups.

**Parameters:**

oldname — the name of the previously existing file to be renamed.

newname — the new name to be assigned to the file. This optional parameter, when used with a file name containing * (a name pattern), indicates that the matching names in the current (or specified) directory and all lower subdirectories, should be renamed. Without this option, the rename applies only to files in the current (or specified) directory.

-REPL — This optional parameter causes the target file (newfile) to be replaced, without any prompting, if it already exists. Without this option, you are prompted whether to replace the file; reply "y" or "yes" to replace it, "n" or "no" to not do the rename. You can also enter "yes all" (to replace the file and all further files for this command, without more prompting) or "no all" (to not rename the file, or any further files for this command which already exist, without more prompting). To cancel the command, enter "/cancel".

-SUB — The -SUB option should be used when renaming an entire directory, or an entire userid. See the examples below.

**Examples:**

```
ren work1 work2
```
Renames the file work1 to work2.

```
ren work* try*
```
Renames all file names that begin with "work" to the corresponding names beginning with "try". For example, WORK23 is renamed to TRY23, WORK.ABC to TRY.ABC, WORK to TRY, etc. Only files in the current directory are affected.

```
rename abc\* defg\* -sub
```
Renames subdirectory ABC to DEFG, including all files in lower subdirectories.

```
ren fred:* susan:* -sub
```
Renames all files in userid FRED to userid SUSAN. (This form of the command is used only by system administrators.)

# /REQUEST

This command is used to send a one-line message to the console operator. As the operator may be busy with other activities, please be patient if the operator does not respond immediately. An installation may limit who can use this command. To send messages to other users, use the TELL command or the MAIL Facility.

**Syntax:**

```
/REQUEST    text of message...
/REQ
```

**Example:**

```
*Go
/request this is a sample message
*OK
```

# RN

The RN command invokes the News Reader containing USENET information. USENET is a network available to users for exchanging messages and information of any kind. The news articles are grouped into categories called *newsgroups*. Newsgroups are devoted to a wide variety of subjects. These include politics, programming languages, science, recreational activities, and many, many more.

The RN command provides access to Network News services. Through this interface you can read incoming news from a variety of news groups, post your own news items, follow-up on existing items and send electronic mail directly to the news contributors.

Help is provided once the program is invoked. For more information refer to the *MUSIC/SP Communications Guide*.

**Syntax:**

```
RN [newsgroup]
```

**Parameters:**

newsgroup The name of the news group that you wish to see. If this is omitted a list of news groups is presented.

# ROUTE

The ROUTE command displays or changes the default route destination for the current MUSIC session. By default, the route destination specified in the userid PROFILE is used automatically. If this does not exist then the system default is used. If the destination is omitted from the command, the current destination is

displayed.

**Syntax:**

```
ROUTE destination
```

# SCHED

Invokes the TODO program called SCHED. It allows you to schedule your personal calendar, meeting room, or equipment. See the *MUSIC/SP Mail and Office Applications Guide* for details.

**Syntax:**

```
SCHED        or         TODO 1
```

# SENDFILE

SENDFILE is used to send a copy of a file to another MUSIC user or CMS user on computers that are connected via RSCS (Remote Spooling Communications Subsystem). Unlike MAIL, SENDFILE sends the file free of mail headers and allows record lengths greater than 80. SENDFILE does not currently use the nicknames file (mail directory), nor does it support email domain names. Therefore, you must specify the exact userid and node (system name) of the user to whom you are sending the file. The MAIL program is used to receive send files.

**Syntax:**

```
SENDFILE filename (TO) userid (AT node)
SF
```

**Parameters:**

filename — is any MUSIC file. It can also be a file name pattern identical to that used with the library command.

userid — is either a MUSIC userid or a 1 to 8 character VM userid.

node — is the system name where *userid* is located. This is not a mail domain name, but the true system node name. If you are sending the file to a user on your system, you don't have to specify the *node*. A node name of "*" can and is used to represent the name of your system.

SENDFILEs you receive on your MUSIC system userid are deposited in your mail box. These are easily identified by the subject line which is in the form "Sendfile: filename". Such files are unaltered by the mail facility. You can then copy the file to any MUSIC file you desire.

*Note:* The TO and AT keywords are not required.

**Examples:**

1. This example sends the file "work" to userid ccfp at node mcgillm.

```
SF work to ccfp at mcgillm
```

2. This example sends all files on my userid that begin with the characters "work" to userid ccfp at node mcgillm.

```
SF work* to ccfp at mcgillm
```

# SENDMAIL

This command invokes the SENDMAIL utility program which is the fast-track method to send a piece of mail. SENDMAIL requires that the text of your message already exist in a file.

**Syntax:**

```
SENDMAIL TO(user1) SUBJ(subject) FILE(filename)
```

The keywords available for use with this command are identical to those keywords used for the MAIL program SEND command. Here are some examples of SENDMAIL commands:

```
 SENDMAIL TO(BOSS) SUBJ(MEETING MONDAY) FILE(MEETING.1)
 SENDMAIL TO(JOE) CC(KATHY) SUBJ(VACATION) FILE(BERMUDA) NOACK
 SENDMAIL TO(JOE,KATHY) SUBJ(CONTRACT) FILE(LAW1)
 SENDMAIL TO(/GROUP) SUBJ(COURSES) FILE(MATH101)
or
 /INC *COM:SENDMAIL
 TO(BOSS) SUBJ(THIS IS A VERY LONG SUBJECT THAT GOES ON AND ON) -
 FILE(MEETING.1)
```

For more information type "/help sendmail".

# SHOPAN

This command displays a panel file without changing it. Each accessible field is numbered and filled with a ruler.

**Syntax:**

```
SHOPAN filename
```

# SHOWPFK

This command lists the program function key definitions for command mode. By default the definitions are as follows:

F1 HELP - invokes the help facility
F4 ADD - adds a new session
F5 DELETE - deletes an extra sessions
F7 PREVIOUS - goes to previous session
F8 NEXT - goes to next session
F12 RETRIEVE - echos text from the command area

**Syntax:**

```
SHOWPFK
```

# /SKIP

This command may be used to skip over unwanted lines of output. This command is effective only when the workstation is in break (attention) mode. The ATTN or BREAK key on the workstation can be used to place the workstation in break mode. Use the PA2 key on a 3270-type workstation.

When a 3270-type workstation is in break mode (**Attn** in the lower right corner of the screen), pressing PA1 has the same effect as entering /SKIP ALL. Thus, to skip all remaining output when in MORE status, press PA1 twice (the first one causes break mode and the second one does the skip).

**Syntax:**

```
/SKIP    [n  ]
/SK      [ALL]
```

**Parameters:**

n is the number of lines of output to be skipped. If *n* is larger than the number of lines of output up to a conversational read or the end of the output, the system skips all lines up to that point. *n* must be positive. If *n* is not specified, no lines are skipped and output resumes. The value for *n* must be 9999 or less (i.e. 1 to 4 digits).

ALL all output lines up to the next conversational read or end-of-job are skipped. A call to system subroutine STOPSK will have no effect.

**Example:**

```
*Go
exec hello
*In progress
```

```
MUSIC

(Multi-User System for Interactive Computing)

(ATTN or BREAK key pressed)
/skip 4
from your workstation.(ATTN or BREAK key pressed)
/skip 100
*End
*Go
```

# SORT

This command is used to sort the records in a file according to a single control field within each record. The system subroutine DSORT is used. The sorted data replaces the original file or a second specified file. The parameters on the command are separated by blanks or commas. (See the topic "Sorting" in *Chapter 10. Utilities* for other sorting programs and subroutines.)

The SORT command is equivalent to running the following job:

```
/PARM filnm1 etc.
/INCLUDE SORT
```

Units 3 and 4 are sort work files, 300K each, which are needed by the DSORT subroutine. The size of the work files may be increased if necessary, by adding overriding /FILE statements to the above job:

```
/FILE 3 NAME(&&TEMP) NEW DELETE SPACE(nnnn)
/FILE 4 NAME(&&TEMP) NEW DELETE SPACE(nnnn)
```

Execution mode, then command mode.

**Syntax:**

```
SORT filenm1 [filenm2][-REPLACE][-NOMSG][n-m][-A][-CH][-DELDUPS]
                      [-R      ][-NOM  ][n  ][-D][-BI]
                                             [-FI]
                                             [-FL]
                                             [-ZD]
                                             [-PD]
                                             [-DA]
                                             [-CI]
```

**Parameters:**

filenm1 The name of the file containing the data to be sorted.

filenm2 The name of the file where the sorted data is to be stored. If *filenm2* is omitted, the sorted data replaces the original file *(filenm1).* If the target file already exists, you are prompted for permission to replace it (unless the -REPLACE option is used). Answer Y or YES to replace the existing file. Any response not starting with "y" stops the program without storing the sorted data.

-REPLACE Causes the target file to be replaced without prompting, if it already exists. Prompting is never done in a batch job. Abbreviations: -R, -RE, -REP, -REPL, etc.

-NOMSG Suppresses information messages during the sort. Error messages are still put out. Abbreviation: -NOM

n-m Defines the starting column number (n) and ending column number (m) of the sort control field within each data record. This is the field on which the records are sorted. n must be from 1 to 4096. If the field is longer than 256 characters, only the first 256 characters are compared. The default sort control field is the entire record. When -*m* is not specified, *n* defines the starting column number of the sort control field, and it is assumed to extend to the end of the record (to a maximum of 256 characters).

-A Records are sorted into ascending (increasing) order. This is the default.

-D Records are sorted into descending order.

-xx Specifies the type of the sort control field. The default type is -CH (character). The possible types are:

- -CH Character.
- -BI Binary (same as -CH).
- -FI Fixed point.
- -FL Normalized floating point.
- -ZD Zoned decimal. This can be used to sort on a field containing a right-justified unsigned decimal number with leading blanks (e.g. as produced by Fortran I format).
- -PD Packed decimal.
- -DA specifies the sort field to be a 7-character date field of the form DDMMMYY (i.e. 01JAN90).
- -CI Case-ignore character field. This is similar to -CH, except upper and lower case characters are considered the same when comparing. For example, a field containing "Fred" is considered equal to a field containing "FRED".

-DELDUPS This option deletes output records that have exactly the same sort control field as the previous output record.

**Examples:**

```
sort master.file new.master

sort my.data -r -nomsg
```

The following commands produce a list of your files, in decreasing order by file size (columns 33 to 37 of the LIBRARY output):

```
library * f s
sort  @lib -r 33-37 -d -zd
list  @lib
*Go
```

```
sort @lib -r 33-37 -d -zd
*In progress
SORTING COLUMNS   33 THRU   37
SRT000  BEGIN SORT.  RECORD LENGTH = 80, AREA =64000
SRT000  RECORD COUNT =     421
SRT000  NORMAL END OF SORT
*End
*Go
```

Example using -DELDUPS option:

Input data (FILE1):
```
ABCX
ABCD
ABCE
AAAA
ABXX
ABXY
```

Command: **`sort file1 file2 1-3 -deldups`**

Sorted data:
```
AAAA
ABCX
ABCD
ABCE
ABXX
ABXY
```

Output data (FILE2):
```
AAAA
ABCX
ABXX
```

# /STATUS

This command is used to get information about the user's workstation and the system. See the example below for details. (The service unit number quoted does not include the charge for I/O to the workstation and the surcharge for regions over 108K.)

**Syntax:**

```
/STATUS
/ST
```

**Example:**

```
*Go
status

1993/04/29       21:37
Userid         USER000
Tnum/Port       66/0A1
Connect time     01:38
Service Units       26
116 Users on MUSIC
```

# SUBMIT

Submit a file or group of files to batch. A complete description of the SUBMIT command is given in *Chapter 3. Using Batch* of this guide.

**Syntax:**

```
SUBMIT filename1 [filename2] ... [filenamen] [parameters]
SUB
```

**Example:**

```
*Go
submit file1
*In Progress
15 RECORDS SUBMITTED.
*End
*Go
```

# SUMMARY

This command can be used to produce a brief summary of the contents of a specified file.

All lines in the file which begin with a slash (/) are displayed. All lines which begin with a 12-2-9 punch (hexadecimal 02) are identified as object decks, and the total number of records in the file is displayed.

**Syntax:**

```
SUMMARY  filename
SUMRY
SUM
```

**Parameters:**

filename is the name of the file.

**Example:**

```
*Go
sumry hello
*In progress

LINE    1 /FILE 9 RDR
        2 /INC SCRIPT
       35 /CANCEL

       89 records counted.

*End
*Go
```

# SYSDATE

This command displays the current date and system level.

**Syntax:**

```
SYSDATE
```

**Example:**

```
*Go
sysdate
*In progress
  FRI MAY 14, 1993     1993/134     18:25:44.19
  MUSIC IPL'd at  7:50 today  (14 May 1993)
  Nucleus level: V24 13MAY93
  MUSIC level:  Version 2   Release  4   Service  0   0
```

# TAG

This command is used to display the tag information associated with a file or to assign new tag information to a file.

**Syntax:**

```
TAG  filename [ tag info ]
```

**Parameters:**

filename — the name of the file whose tag is to be displayed or assigned.

tag info — optional 64 bytes of tag information to be assigned to the file. If omitted, the file's existing tag is displayed.

**Example:**

```
*Go
tag work1 sample file to show the use of tag
*In progress
TAGGED
*End
*Go

tag work1
*In progress
TAG IS: SAMPLE FILE TO SHOW THE USE OF TAG
*End
*Go
```

# TEDIT

This command is used to invoke the Editor program to edit a file. This command differs from the EDIT command only in that a TEXT SCRIPT Editor command is implied and tab characters are not translated to an appropriate number of blanks. This means that any lower case letters typed in are NOT be translated to upper case. For details about the Editor refer to *Chapter 7. Using the Editor* of this guide.

**Syntax:**

```
TEDIT [name] [NEW] [LRECL(n)] [fmt] [RO] [NOLOG] [UIO] [MAX(n)]
TED                [n       ]
```

**Parameters:**

See EDIT command for parameter descriptions.

# TELL

This command is used to send a single line message to another user who is currently signed on to the system. The message text is immediately displayed on the receivers screen. If the user is not signed on or has suppressed messages (/MESSAGES OFF) nothing is sent. The MAIL Facility can be used to send longer messages and does not require the receiver to be signed on when the message is sent.

Intersystem tell messages are allowed if your site is connected to BITNET and your system supports this feature.

**Syntax:**

```
TELL userid [message-text]
TELL userid@systemid [message-text]
```

**Parameters:**

userid The userid of the person who is to receive the message. The message is sent to each active session using that userid.

userid@systemid
The BITNET address of the person who is to receive the message. The systemid must be a valid BITNET node name. Check with your installation to see if you are connected to BITNET and whether the intersystem tell is supported.

message-text The message test to be displayed on the screen. The case of the text is preserved. A prefix is added to the message indicating who the sender is.

**Example:**

```
*Go
tell ax01 Hi there, What about lunch.
*In progress
*End
*Go
```

# TELNET

This command is used to invoke access other computers connected via TCP/IP to the Internet. A list of Internet addresses that allow anonymous access is supplied. For a brief description of TELNET see *Chapter 10 - Utilities* under "FTP and TELNET from MUSIC". For full details refer to the *MUSIC/SP Communications Guide*.

**Syntax:**

```
TELNET [internet address]
```

# /TIME

This command may be used to determine the time of day and how much computer time has been used up to the present time in the job which is presently running. It is particularly useful if it is suspected that a program may be in a loop. The command may be entered after attention or break has been pressed to put the workstation in break mode. In this case the execution time will represent that used so far by the current job. If the command is entered from command mode the execution time is the total used by the last job to run (the ? abbreviation cannot be used in this case). The command cannot be used when the workstation is in reading mode.

The execution time is displayed in service units. If an asterisk (*) follows the time it indicates that your job was getting service at the instant the command was processed.

Use from break mode does not effect the execution of the program.

**Syntax:**

```
/TIME
/T
?
```

**Example:**

```
*Go
list sample
*In progress
      READ(9,*) A
      B=SQRT(A)
      WRITE(6,*) A,B
      CALL EXIT
      END
*Go
sample
*In progress
(ATTN or BREAK key pressed)

/time
21:54   Job time 0.2  SU
```

# TODO

This command invokes the Time, Office, and Documentation Organizer (TODO) facility. This facility allows you to access various components of the MUSIC system through a selection menu. This menu consolidates the most frequently used programs for an office environment. For example, access to SCRIPT (word processing program), MAIL (electronic mail), SPELL (spell checking), etc. Help is provided once the facility is invoked. For complete information refer to the *MUSIC/SP Mail and Office Applications Guide*.

**Syntax:**

```
TODO [n]
```

# TREE

Displays your directories graphically and allows you to select directories by tabbing to each name.

**Syntax:**

```
TREE
```

Help is available once the command is entered.

# TUT

This command invokes the TUTORIAL facility for learning programming languages.

**Syntax:**

```
TUT
```

Help is available once the command is entered.

# /USERS

This command is used to find out how many users are active (signed on the system) at the present time.

**Syntax:**

```
/USERS
/U
```

**Example:**

```
*Go
users

062 Users on MUSIC
```

# VER

Displays the current MUSIC version and release numbers.

**Syntax:**

```
VER
```

# VIEW

This command invokes the VIEW utility program for viewing files of any record length, type, or 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.

For full information including command descriptions, refer to "VIEW" in *Chapter 10. Utilities.*

**Syntax:**

```
VIEW filename
```

See also EVIEW.

# VM

The MUSIC/SP Passthru Facility allows you to create a VM session directly from your MUSIC session. The VM command creates the VM session and allows you to connect to CMS, TSO, VSE or another MUSIC machine. A simple escape sequence allows you to issue commands directly to the local session. The facility also supports file transfer between the local session and the remote session using the standard IBM 3270 file transfer protocol.

## Creating a VM session

When the command VM is issued a VM session will be created. You must now enter the appropriate commands to connect to CMS or whatever application is required. You cannot dial back into the local MUSIC system.

MUSIC still controls your workstation. Messages from other users or the operators will still be displayed. Because of conflicts with CMS and TSO the full screen escape key (PA2) is disabled by the VM command, but multi-session commands can still be issued if prefixed by a percent sign (%).

## Issuing Local Commands

The percent sign has been defined as an escape sequence that causes the command to be executed on the local system, rather than on the session provided by Passthru. The following commands are not passed to MUSIC but are processed directly by the VM program itself.

```
%RECEIVE      - Receive a file from a remote session
```

```
%SEND         - Send a file to a remote session
%END          - Terminate remote session.
%QUIT         - Terminate remote session.
```

Care must be taken when entering "local commands" (%RECEIVE, %SEND, %END, %QUIT). MUSIC looks for the local command at the start of the 3270 data stream, and expects a blank or no data to follow the command keyword. If other screen fields are modified, or marked by the system as modified (as in the VM logon screen), MUSIC may not recognize the local command. The command is recognized in most cases if you follow these rules:

1. Type the local command in the FIRST modifiable field.
2. Type a blank after the local command. (This blank is not always needed, but it is needed on the VM logon screen.)

## File Transfer

The SEND and RECEIVE commands are used for file transfer. File transfer is only possible if the system running the remote session has the IBM 3270 File Transfer program installed. Usually this program is used to transfer files between a mainframe and a PC. The local session running VM command plays the role of the PC in this file transfer operation, however since it is running on an IBM mainframe no EBCDIC conversion is required. Both commands have the same format.

**Syntax:**

```
SEND    local-file remote-file [options]
SEND    >list-file
RECEIVE local-file remote-file [options]
RECEIVE >list-file
```

**File Transfer Parameters**

| | |
|---|---|
| local-file | The name of the local file involved. |
| remote-file | The name of the remote file. |
| options | Optional parameters. If the remote session is on CMS these must be preceded by an open brace "(". The options are discussed below. |
| list-file | The file name of a local file containing a list of file transfer parameters. This is useful if a group of files is to be transferred at the same time. |

**Options**

| | |
|---|---|
| APPEND | Append to the file on the remote session. |
| ASCII | Convert file to ASCII. This is not useful in host/host file transfer. |
| CRLF | Use carriage return/line feed characters as record separators. This should always be specified if you want the record oriented characteristics of the file preserved. It should never be used when transferring binary data that may already contain such sequences. |

There are other options that are dependent on which remote system you are sending to. Consult documentation on the IBM 3270 File Transfer Program for details.

**File Transfer Notes**

The standard procedure for the 3270 File Transfer program is to replace existing files without prompting the user. This is also true in this implementation.

To transfer binary data that also has logical records (object modules), pre-allocate a file of the appropriate record length on the target system and then do the file transfer. Do not specify the CRLF option.

The current implementation supports record lengths of up to 2048 bytes.

# WEB

This commands invokes the Web line-mode browser for accessing Internet Web sites. Help is available once the program is invoked.

**Syntax:**

```
WEB {url address}
```

For example,

```
WEB http://musicm.mcgill.ca
```

# WHOAMI

This command displays userid information and TCB number for the current session.

**Syntax:**

```
WHOAMI
```

**Example:**

```
*Go
whoami
*In progress
You are userid CCGW000 on virtual machine MUSIC
Session id (TCB number) is 17
File ownership id is CCGW
Subcode is 000
```

# /WINDOW

This command instructs MUSIC to display only certain portions of each line directed to the workstation. For example, "/WINDOW 5,50" displays only the portion of each line that normally displayed in columns 5 through 50. The characters are displayed starting in the first position of the output line.

This command is particularly useful when displaying long lines of output on an IBM 3270 terminal or other workstation which has a short line length.

The effect of this command is removed when the workstation next enters command (*Go) mode.

If the column numbers are not respecified, they will default to the ones last given.

**Syntax:**

```
/WINDOW  [m,n] [,OFF ]
/WI
```

**Parameters:**

m,n — Specifies the start and ending column numbers of the window. The first output position is called number 1. (Normally when your program displays on the workstation, the first character position is used for a carriage control character that is not displayed and it is therefore not counted as output position 1.)

OFF — Turns off the window feature.

**Example:**

```
*Go
list sample
*In progress
1234567890ABCDEF
*End
*Go
window 10,12
*OK
list sample
*In progress
 0AB
*End
*Go
```

# XTMUS

Transfers PC files to MUSIC. You must be using PCWS on your PC to connect to MUSIC. For complete details, refer to the *MUSIC/SP Personal Computer WorkStation User's Guide*, or type "HELP XTMUS".

**Syntax:**

```
XTMUS src [dest] [-BIN] [-REPL] [-APP] [-COMP] [-Fx] [-n]
                 [-B  ] [-R   ] [-A  ] [-C   ]

                 [-MON] [-SWITCH] [-NOMSGS]
                 [-M  ] [-S     ] [-N     ]
```

# XTPC

Transfers MUSIC files to the PC. You must be using PCWS on your PC to connect to MUSIC. For complete details, refer to the *MUSIC/SP Personal Computer WorkStation User's Guide*, or type "HELP XTPC".

**Syntax:**

```
XTPC src [dest] [-BIN] [-REPL] [-APP] [-COMP]
                [-B  ] [-R   ] [-A  ] [-C   ]

                [-MON] [-SWITCH] [-NOMSGS]
                [-M  ] [-S     ] [-N     ]
```

# ZEROCNT

The ZEROCNT command resets a file's usage count to 0 by archiving it to a temporary file, then restoring it. The file's dates are also reset.

**Syntax:**

```
ZEROCNT  {filename} {pattern} {<listname}  *
```

**Parameters:**

| | |
|---|---|
| | filename is the name of the file. |
| pattern | is the file specification including wild characters * and/or ?. Note that a pattern only applies to files in the current directory. |
| <listname | is the name of the file containing a list of file names. |
| * | give the count for all your files that have the CNT attribute. |

# Chapter 6. MUSIC Job Control Statements

# Job Control Statements - Overview

Job control statements are used to control program execution. They define the environment in terms of time and storage limits, define the files used, define the type of process to be run and set optional parameters. The job control statements are processed only when the file containing them is executed. Errors in statement syntax will only be noted at this time. The system processes all control statements up to and including the /LOAD statement. After that the process that is loaded takes over control of reading the input from the file. For this reason statements that apply to the system (/SYS or /FILE) must come before the /LOAD and those that set options for the loaded process (/OPT or /JOB) must come after the /LOAD. The following outlines the order of statements in a typical job.

```
/PARM
/SYS
/FILE
/LOAD
/JOB
/OPT
. . . . .      (Source Language if applicable)
/DATA
. . . . .      (Data records if applicable)
```

Some statements such as /INCLUDE, which includes statements from another file, can appear anywhere in the sequence. Others such as /INFO and /PASSWORD must appear first if used.

All job control statements begin with a slash (/), followed by the statement verb. One or more blanks separate the statement verb from variable parameters (which must begin before column 16). Variable parameters are separated by blanks or commas.

## Conventions

The following conventions are used in this chapter to describe Job Control statements:

1. Upper case letters and punctuation marks represent information that must be coded exactly as shown.

2. Lower case letters and words are generic terms representing information that must be supplied. That is, a substitution must be made when coding a parameter or option so represented.

3. Information within square brackets [ ] represents an option or choice of options that may be included or omitted, depending on requirements.

4. In the examples, information typed by the computer is shown in upper case, and information typed by the user is shown in lower case.

5. Underlined parameters are the default values.

6. The only statement abbreviation allowed from batch is /INC for /INCLUDE.

The remainder of this chapter gives a description of all Job Control Statements.

## /COM

This statement is used in a file to indicate a comment. These statements are ignored as long as they appear before a /LOAD statement.

**Syntax:**

```
/COM text
```

*Note:* If /COM is used between a /FILE statement and a /ETC statement, the /ETC statement is also ignored.

## /DATA

This statement is used as an indication that all lines following are data lines to be processed during program execution, and are not to be processed by a compiler or loader program. It is not required unless data lines are to be read during program execution from the file.

**Syntax:**

```
/DATA
```

**Example:**

```
*Go
list dat123
*In progress
25.
36.
49.85
*End
*Go
```

```
list sample
*In progress
     1 READ(5,2,END=10)A
     2 FORMAT(F10.2)
       B=SQRT(A)
       WRITE(6,3)A,B
     3 FORMAT(' THE SQUARE ROOT OF',F6.2,' IS',F5.2)
       GO TO 1
    10 STOP
       END
/DATA
/INCLUDE DAT123
*End
*Go
sample
*In progress
MAIN   = 0001A4
003658 BYTES USED
EXECUTION BEGINS     1.4S
THE SQUARE ROOT OF 25.00 IS 5.00
THE SQUARE ROOT OF 36.00 IS 6.00
THE SQUARE ROOT OF 49.85 IS 7.06
STOP         0
*End
*Go
```

# /END

This statement is used as a delimiter to indicate the end of a batch job.

**Syntax:**

```
/END
```

# /ETC

This statement is used as a continuation line for the /ID and the /FILE lines. See the /FILE statement writeup below for its usage with that statement.

The /ETC statement can be used as a continuation line for the batch /ID statement. In this case, it is used solely for comments to the operator in regards to where or how to return your printed output. See *Chapter 3. Using Batch* for more details.

**Syntax:**

```
/ETC  [information]
```

# /FILE (General Overview)

Programs usually refer to files and input/output (I/O) devices through *ddnames* (data definition names) or unit numbers. The /FILE statement provides the connection between the program and the actual file by relating the symbolic ddname or unit number to the physical file or device. /FILE statements are processed at execution time, and may be changed from one run of the program to another. In this way a single program can be used to process data from a variety of sources without having the change the program.

Suppose, for example, a program reads from unit 8 and writes to unit 9. It can perform a number of different functions depending on how units 8 and 9 are defined through /FILE statements. If they are both defined as files, the program is copying one file to another. If unit 8 is defined as a file and unit 9 is a printer, the program is printing a file.

The /FILE statement gives the name of a file and some of its characteristics. The program communicates with this file through a logical unit number or data definition name (ddname). (Fortran programs can use logical unit numbers, COBOL programs can only use ddnames.) When the user program does I/O through the logical unit number or ddname, the actual read and write action will be done to the corresponding I/O device or file.

MUSIC allows a user to define a logical unit number from 1 to 15 inclusive. A single /FILE statement cannot refer to two or more I/O devices or files. No duplicate logical unit number or ddname can be defined within one job. The type of I/O devices or files that can be defined will be detailed later as they are discussed separately under three major categories: files, UDS files, and miscellaneous. Additional parameters such as the record size, blocksize, record format, etc. of the file being defined will also be discussed in detail under each major category.

All jobs have the following default logical unit assignments:

```
/FILE  5  RDR
/FILE  6  PRT
/FILE  7  PUN
/FILE  9  TERM
/FILE 10  HOLD
```

The meaning of the above /FILE statements will be discussed later under the 'miscellaneous' category.

**Syntax:**

**General Description**:

```
/FILE  u    type  ...
/FILE  ddn  type  ...
```

The syntax of the /FILE statement requires the logical unit number (u) or the ddname (ddn) to be the first parameter defined and separated by at least one blank from the statement word /FILE. The second parameter that must be defined is the type of I/O device or file. Additional parameters can then be defined in any order. Parameters must be separated by one or more blanks or commas. Blanks will be used to separate parameters in the following discussion. Each parameter is specified in the form of KEYWORD or KEYWORD(SUBPARAMETER), for example, RDR, SPACE(10).

The /FILE statements are placed at the beginning of a job and must occur before any /LOAD statements or they will not be processed. The /FILE control line cannot exceed 80 characters in length. If more space is

required, the /FILE line may be continued on a /ETC control line. The parameters may be split between a /FILE and a /ETC line at any place where a blank or a comma is valid, except that a subparameter must not be split between lines.

A single job is limited to a maximum of 4 UDS-type files (including tape files). Some processors (for example PL/I) allow a maximum of only 3 user-defined UDS files. Up to 14 files can be in use at any one time. Files defined via the /FILE statement are considered in use during the entire job. Files referenced by /INCLUDE statements or opened dynamically during the execution of a job count against this limit only when they are in use (i.e. opened).

# /FILE (Files)

Files can be created and/or deleted in one job. They could be created in one job and used in another at some later time. In order to be able to refer to the same file again later, the user must give a name to a newly created file. Besides the name, the user can also specify a number of items such as the record length, record format, or the attributes of the file. These items are assigned to a file at the time it is created and cannot be modified afterwards.

Each userid contains a limit to the amount of space that can be allocated to any one file. This is done to guard against abusive use of disk space and in fairness to the other users who may also want to use some of the same space. Should the space limit pose any difficulties, the user should go to the MUSIC Systems Administrator to request a bigger space limit.

A charge is normally made for the space that you allocate for your Save Library. This charge is based on the amount of space occupied and the length of time the file exists.

MUSIC features a facility to automatically preserve the integrity of your files by allowing them to be private, public *execute-only*, or public *read/write*. Furthermore, the system automatically performs an enqueue operation to ensure that no two jobs are accessing the file in a manner that may affect each other. The various types of enqueue protection will be detailed under the *disp* parameter discussed below. The message `SAVE FILE IS IN USE...TRY AGAIN LATER` will be issued if the enqueue detects disallowed multiple use of the file.

**Syntax:**

```
Files

/FILE  u    Name(fn1) [disp] [RLSE  ] [LRecl(l)] [RECFm(fm)]
       ddn   PDS(fn2)        [NORLSE] [RSIZe(l)]

             [SPace(ps)] [SECsp(ss)] [MAXSP(ms)] [attrib] [NRec(n)]
```

**Parameters:**

u — specifies the logical unit number by which the file will be referenced. It must be a number from 1 to 15 inclusive. No duplicate logical unit number can be defined in one job.

ddn — specifies the step name and data definition name (ddname) by which the file will be referenced. If only a ddname is specified, the ddname will be applied to both the COMPILE step and the GO step. If both the step name and ddname are specified, they must be specified in

the form *stepname.ddname*. The valid stepnames for the COMPILE step are COMP, FORT, PLI, COB, and ASM. The only valid stepname for the GO step is GO. For example, FORT.SYSIN, GO.DATA, FILE1. No duplicate ddname can be defined in one job.

Name(fn1) specifies the name of the file involved. Refer to *Chapter 4. File System and I/O Interface* for the naming convention of these files. If NAME(&&TEMP) is specified, the file created will be a temporary one which means that it will be discarded when the job is finished. The parameters NEW DELETE are automatically assumed when NAME(&&TEMP) is used.

PDS(fn2) specifies the name of a group of files. VS Assembler uses this parameter to define macro libraries. COBOL (COPY verb) and PL/I (%INCLUDE statement) also use this parameter to incorporate source statements. A ddname, and not a logical unit number, must be specified if this parameter is used. *fn2* is specified in the form of, for example, ABC.*.X. If a member name, for example, M1 is referenced by the specified ddname in the user program, the resulting file involved will be ABC.M1.X. Furthermore, the PDS parameter allows the specification of multiple groups. For example, if PDS(ABC.*.X,MN.*.YZ) is specified, the file ABC.M1.X would be searched for first and if not found, the file MN.M1.YZ will then be tried.

disp specifies the disposition of the file. It indicates the current status of the file and what MUSIC will do to it after the job ends. Items that can be specified are:

NEW indicates that the current status of the file is a new one and space will be allocated for it. The user can write to the file. An error message will be issued if the named file actually does exist in the Save Library. However, if NEW(REPLACE) is specified, the existing file will be purged (deleted) from the Save Library and a new file will be created under the same name. REPLACE can be abbreviated as REP or REPL. Only the current job is allowed to access the file while it is running.

OLD indicates that the current status of the file is old, meaning that it already exists in the Save Library. The user can write to the file. An error message will be issued if the named file actually does not exist. However, if OLD(CREATE) is specified, a new file will be created if the named file does not exist. CREATE can be abbreviated as CR. Only the current job is allowed to access the file while it is running.

SHR has two meanings. For new files, it means that the file is to be saved so that other users can refer to it by prefixing the creator's userid in front of the file name. For existing (old) files, it indicates that more than one job can access the file concurrently. This also prevents any of the jobs from writing to the file while this job is running. For existing files, SHR is assumed if no disposition is specified.

WSHR indicates that the current status of the file is old and that more than one job can write to it at a time. This feature should be used with caution. Discipline should be provided in the programs to prevent one from destructively interfering with another. While this job is running, other users are allowed, if the file is a public one, to access the file as SHR but not as OLD.

APPend indicates that the current status of the file is old and that when output is written to the file, it will be written starting at the end of the file.

APPONLY (append only) indicates that output will be written to the end of the file, and this is the only activity allowed. No reads may be done. A current status of OLD is assumed. SHR and WSHR may not be used.

KEEP — indicates that when the job is finished, the file is to be kept. KEEP is always assumed unless DELETE is specified or the file name is &&TEMP.

DELETE — indicates that when the job is finished, the file is to be deleted (removed) permanently from the Save Library. The parameter OLD is always assumed when DELETE is used.

DEFault — causes the /FILE statement to have no effect if some previous /FILE statement has been given for the same logical unit number or ddname. This feature is useful when setting up generalized programs particularly when they are to be used by other users who may want to use their own /FILE statements in place of some default ones you provide.

Some typical disposition combinations are shown below:

```
NEW
NEW(REPLACE)
OLD
OLD(CREATE)
NEW DELETE
OLD DELETE
NEW DELETE DEFAULT
NEW DEFAULT
```

RLSE — specifies that any unused space at the end of the file is to be released. This means that the file is to be made as small as possible to hold the existing data. This is the default unless SHR is specified.

NORLSE — specifies that any unused space at the end of the file is not to be released.

LRecl(l) — specifies the length of each record of the file. The maximum record length is 32760 bytes. The minimum record length is 1 byte. If omitted, a record length of 80 bytes is assumed. This parameter is only used when the file is being created. For FORTRAN sequential I/O, the maximum logical record length is 133 (unless the system subroutine BIGBUF is called).

RSIZe(l) — same as LRECL.

RECFm(fm) — specifies the record format of a file. The valid record formats are F (fixed), FC (fixed compressed), V (variable), VC (variable compressed), and U (undefined). Refer to *Chapter 4. File System and I/O Interface* for an explanation of the various record formats. The default is FC. This parameter is only used when the file is being created.

SPace(ps) — specifies the primary space allocation of the file. ps is specified in units of K (1024) bytes. The *K* should be omitted from the parameter. The default is 32K bytes. This parameter is only used when the file is being created.

SECsp(ss) — specifies the secondary space allocation of the file. ss can be specified in units of K bytes or in units of percentage of the current space allocation. For example, SECSP(10) means that the secondary space allocation is 10K bytes, and SECSP(30%) means that it is 30% of the current space allocation. The default is 50% of the current space allocation. This parameter is only used when the file is being created.

MAXSP(ms) — specifies the maximum space limit of the file in units of K bytes. The default value is the maximum allowable space that can be allocated to the user for each file. This parameter is only used when the file is being created.

attrib — specifies the attributes of the file and is used when the file is being created. The valid attributes are shown below. Some users may be restricted from specifying the SHR or PUBL option.

- PRIV — indicates that the file is *private.* This means that it can only be used by the *owner* (the user who signed on to MUSIC with the particular userid which was used to create the file). This is the default attribute.
- PUBL — indicates that the file is *public.* This means that the file is in the common library and that any user can access it. However, only the *owner* can write to or delete the file.
- SHR — indicates that the file can be read by other users but that its name is not to be stored in the common library. Other users must refer to this file by prefixing the owner's userid in front of the file name as in the example userid:PROG. Only the *owner* can write to or delete the file.
- COM — indicates that the file is *private* and it is stored in the common library index.
- XO — indicates the file has the *execute-only* attribute. A file of this type cannot be listed, displayed, or inserted. For complete details refer to the XO parameter of the MUSIC command /SAVE in *Appendix A. /Input Mode*.

Some typical attribute combinations are shown below:

```
PRIV
PUBL
COM
PUBL XO
```

NRec(n) — specifies the maximum number of records in the file. If the parameter SPACE is not used, the number of records, in conjunction with the specified record length (RSIZE or LRECL), is used to determine the approximate amount of space allocated to the file, by multiplying the two numbers. However, if the SPACE parameter is used, the amount of space allocated will be solely determined by the SPACE parameter and NREC will be ignored. This parameter is only used when the file is being created.

**Examples:**

```
/FILE  3  N(MYFILE)
/FILE  GO.OUTFIL  NAME(PGM1.OUT)  APPEND
/FILE  FT01F001  N(DATA.ONE)  NEW  SP(10)
/ETC   SEC(5)  RLSE  PUBL  LRECL(80)
/FILE  GO.DATA1  PDS(DATA.*.X)
```

```
*Go
list sample
*In progress
/FILE 10 NAME(XXX) OLD
      DO 50 I=1,100
      J=I**2
      WRITE(10,30)I,J
30    FORMAT(I3,3X,I5)
50    CONTINUE
      .
      .
      .
      ETC.
```

*Note:* For FORTRAN programs and others using the FORTRAN/MUSIC interface (COBOL, PL/I, and ASM programs do not use this interface), the default maximum record size is 133 bytes for sequential I/O. It can be increased by a call to the subroutine BIGBUF. If FORTRAN unformatted I/O is used, allowance for a four-byte control word must be included.

# /FILE (Temporary UDS)

Temporary Disk UDS files are those used during a single job and then discarded. The total number of temporary and permanent UDS files defined in one job cannot exceed four. In order to fairly share the common pool of temporary disk space, MUSIC sets a limit to the total amount of temporary disk space any one job can use at any one time through all the user's /FILE statements. This limit is roughly equivalent to the space required to hold 48,000 80-byte or 32,000 128-byte records. If this proves a limitation, then you can use a permanent UDS file and specify the disposition NEW DELETE on it to make it behave like a temporary UDS file.

Some compilers and loaders also require some of this temporary space in order to handle your job. Generally they will try to use what is left over after your /FILE statement definitions, though this may not be sufficient in some unusual conditions.

**Syntax:**

**Temporary Disk User Data Set (UDS) Files**

```
/FILE u    UDS(&&TEMP) [NRec(n)] [LRecl(l)]
      ddn                        [RSIZe(l)]
```

**Parameters:**

u — specifies the logical unit number by which the temporary UDS file will be referenced. It must be a number from 1 to 15 inclusive. No duplicate logical unit number can be defined in one job.

ddn — specifies the step name and data definition name (ddname) by which the temporary UDS file will be referenced. Refer to the writeup of the /FILE statement for the files for complete details.

UDS(&&TEMP)

specifies that the file involved is a temporary UDS file.

NRec(n) specifies the maximum number of records that you will use in the file. If this parameter is omitted, a number of records equivalent to 800 128-byte records will be used.

LRecl(l) specifies the length of each record of the temporary UDS file. This can be any number of bytes between 20 and 512 inclusive. If omitted, a record length of 128 bytes is assumed. For FORTRAN sequential I/O, the maximum logical record length is 133 (unless the system subroutine BIGBUF is called).

RSIZe(l) same as LRECL.

**Examples:**

```
/FILE  5  UDS(&&TEMP)  NREC(500)
/FILE  XFILE  UDS(&&TEMP)  NREC(1000)  RSIZ(80)
/FILE  15  UDS(&&TEMP)  LRECL(100)
```

*Notes:*

1. For FORTRAN programs and others using the FORTRAN/MUSIC interface (COBOL, PL/I, and ASM programs do not use this interface), the following comments apply:

   The default maximum record size is 133 bytes for sequential I/O. (Sequential I/O is the normal access technique.) It can be increased by a call to the subroutine BIGBUF. If FORTRAN unformatted I/O is used, allowance for a four-byte control word must be included. If the data set is to be used for FORTRAN direct access, the DEFINE FILE statement in the FORTRAN program may specify any record size from 1 to 512 bytes. In this case the value of LRECL is used only to calculate how much disk space must be allocated.

   This also applies to the permanent UDS files.

2. There is another form of the /FILE statement which can be used to define temporary UDS files. The syntax of this form is:

   /FILE DISK=(u,NREC=n),RSIZ=r

   This form is used by the older versions of MUSIC. Users are NOT encouraged to use this form of the /FILE statement as it is not as powerful as the new form and will not be supported at some later time. Refer to the User's Guide of older versions of MUSIC for a description of the parameters used.

# /FILE (Permanent UDS)

Permanent disk UDS files are those that can be created in one job and used in another at some later time. In order to be able to identify the same file again later, you give the file a name called the data set name (dsn). You also choose the disk volume that is to hold your file. This allows you to separate your UDS files on different volumes to optimize job time. MUSIC1 is a valid disk volume name. Check with your installation about other disk volume names available to you.

The parameters NREC and RSIZE (or LRECL) are only given when you create the file. When you use the file later on, the same numbers will apply as when the file was created.

Each userid contains a limit to the amount of space that can be allocated to any one file. This is done in fairness to the other users who may also want to use some of the same space. The amount of space that you may

allocate can be determined by running the MUSIC PROFIL utility program. The MUSIC Systems Administrator can allocate specific larger files for you if you have this requirement.

A charge is normally made for the space that you allocate for your permanent UDS files. This charge is based on the amount of space used and the length of time the file exists.

MUSIC features a facility to automatically preserve the integrity of your UDS files by allowing them to be private, public *read-only*, or public *read/write*. Additionally, the system automatically performs an *enqueue* operation to ensure that no two jobs are accessing the file in a manner that may affect each other. The various types of enqueue protection are detailed under the *disp* parameter discussed below. The message `DATA SET IN USE...TRY AGAIN LATER` will be issued if the enqueue detects disallowed multiple use of the file.

**Syntax:**

```
Permanent Disk User Data Set (UDS) Files

/FILE u   UDS(dsn) [NRec(n)] [LRecl(l)] VOLume(v) [disp]
      ddn                    [RSIZe(l)]
```

**Parameters:**

u — specifies the logical unit number by which the permanent UDS file will be referenced. It must be a number from 1 to 15 inclusive. No duplicate logical unit number can be defined in one job.

ddn — specifies the step name and data definition name (ddname) by which the permanent UDS file will be referenced. Refer to the writeup of the /FILE statement for the files for complete details.

UDS(dsn) — specifies the data set name. The dsname field can be up to 22 characters in length. A data set naming convention is established for MUSIC whereby the owner of the file can be identified. There are two types of dsnames:
(1) A dot (.) is used to separate the ownership id from the name. The dsname can be up to 22 characters long. For example: "GEORGEW.TEMP1".
(2) When there is no dot (.) in the dsname then the first 4 characters of the dsname must match your userid. The maximum length of the dsname is 8 characters. For example: "CCGWTMP1". Type 2 is available for compatibility with older versions of MUSIC.

Access to a UDS is governed by the use of an indicator character included in the dsname. This character is in the 1st position after the dot (type 1) or in the 5th position (type 2). The indicator may be one of the following:

$ for a private data set: can be read or written only by the owner. That is, the user code used at sign-on must match the user code in the data set name.

# for a writable public access data set: can be written by anyone.

any other letter or numeric digit for a public *read-only* data set: can be read by anyone but can only be modified or deleted by the owner.

NRec(n) — specifies the number of records in the file. This parameter is only used on the /FILE statement when the file is being created. Refer to the writeup about temporary /FILE

statements for more information about this parameter.

LRecl(l) specifies the record length for files that are being created. It must be between 20 and 512 inclusive. If omitted, the record size is assumed to be 128. This parameter cannot be used to change the record size of an existing file. Refer to the LRECL description under the writeup for temporary UDS files for more information about this parameter. For FORTRAN sequential I/O, the maximum logical record length is 133 (unless the system subroutine BIGBUF is called).

RSIZe(l) same as LRECL.

VOLume(v) specifies the disk pack volume name for this data set. MUSIC1 is a valid volume name. Check with your installation about other volume names available to you.

Normally the volume name refers to a disk pack that is already mounted and accessible at the central computer site. Jobs run from batch can request that special volumes be mounted for the duration of the job. If the volume name given by this parameter is not currently mounted, then MUSIC will assume that you are requesting the mount of a special volume. Your MUSIC Systems Administrator may be of assistance in setting up such a volume. You may be required to specify the general type of disk pack that is to be mounted. This is done through the choice of one of the following numbers: 2311, 2314, 3330, 3340, 3350, 2305, 2305.1, 2305.2. This number is specified as a parameter in the /FILE statement. The form of the parameter is DEVice(d) where d is one of the numbers mentioned above.

disp specifies the disposition of the UDS file. It indicates the current status of the file and what MUSIC will do to it after the job ends. Items that can be specified are the same as those in the /FILE statement for the files with the following exceptions:

1. The REPLACE option of the disposition NEW, i.e. NEW(REPLACE), is not allowed.
2. The CREATE option of the disposition OLD, ie. OLD(CREATE), is not allowed.
3. The dispositions APPEND and APPONLY are not allowed.
4. For new UDS files, you can request that you want this file to be flagged as requiring backup. This is specified by the parameter BACKup. The opposite is NOBACKup which is assumed by the system if BACKUP is not specified.

Refer to the writeup of the /FILE statement for the files for a description of each item.

Some typical disposition combinations are shown below:

SHR
NEW
OLD
NEW DELETE
DELETE
NEW DELETE DEFAULT
NEW BACKUP

**Examples:**

```
/FILE    12  UDS(xxxxABCD)  VOL(MUSIC2)  OLD
/FILE  DATA  UDS(xxxx$PRV)  VOL(MUSIC2)  SHR
/FILE     2  UDS(xxxx#PUB)  VOL(MUSIC3)  NREC(1000)
/ETC  RSIZ(80)  NEW  BACKUP

*Go
list sample
*In progress
/FILE  1  NAME(ABCD)  NREC(35200)  RSIZ(32)
/FILE  2  UDS(USERXXXX)  VOL(MUSIC2)  OLD
       DIMENSION TABLE(8)
       READ(2,10)TABLE
10     FORMAT(8A4)
       WRITE(1,10)TABLE
       .
       .
       .
       ETC.
```

*Notes:*

1. If your job abnormally terminates, it is possible that the system will disregard the KEEP attribute for a NEW file. (This will happen ONLY for NEW files.) If the entire MUSIC system should fail, before your job finishes, it is possible that a file that you had asked to be deleted at the end of your job will not be deleted. To avoid these cases, it is advisable to create and delete your UDS files separately from any long running program that will use them. The following are suggested techniques for performing these creation and deletion operations:

   Creation of UDS Files:

   ```
   /FILE 1 UDS(...) NEW ...
   /INCLUDE UTIL
   $INFO
   ```

   Deletion of UDS Files:

   ```
   /FILE 1 UDS(...) DELETE
   /LOAD IEFBR
   ```

2. There is another form of the /FILE statement which can be used to define permanent UDS files. The syntax of this form is:

   ```
   /FILE DISK=(u,dsname,NREC=n),RSIZ=r,VOL=v,DISP=d
   ```

   This form is used by the older versions of MUSIC. Users are NOT encouraged to use this form of the /FILE statement as it is not as powerful as the new form and will not be supported at some later time. Refer to the User's Guide of older versions of MUSIC for a description of the parameters used.

# /FILE (Tape UDS)

UDS files on magnetic tape are handled in a similar way to those for disk UDS files. Magnetic tape units come in two different versions: 9 and 7 track and each type can record at different densities.

MUSIC automatically handles the blocking of records for disk data sets but for tape you must specify how your records are to be blocked. This allows for compatibility with other operating systems. Normally you would want to use a large blocksize to reduce the amount of tape used as well as reduce the processing time. The block size (BLKSIZE) should be a multiple of the record size (RSIZE). For example, BLKSIZE(800) and RSIZE(80) is permissible.

Magnetic tapes have a special circular ring known as a *write-enable ring*. When this ring is removed from the tape, then no write operations will be allowed on the tape. The ring can be replaced at a later time, should you wish to write on the tape again. MUSIC features an additional write protection through the use of the disposition SHR. In order to write on a tape from MUSIC both the write-enable ring must be in place on the tape and the disposition SHR must NOT be specified.

Some operating systems put special *label* records on the tape. MUSIC does not. Label records, if present, will look to MUSIC just like a file written by a user program. You may skip over the label records if you want by first reading until you get an end-of-file indication. This EOF indication would mean that you are now beyond the label records and that you can start reading the data file that follows. (Some installations use MUSIC to read the label records created by other operating systems when the contents of these records are in doubt.)

**Syntax:**

**Magnetic Tape User Data Set (UDS) Files**

```
/FILE u    TAPE [BLksize(s)  ] [LRecl(l)] VOLume(v) [disp]
     ddn        [BLocksize(s)] [RSIZe(l)]

                [DENsity(d)] [9TRk  ] [TRTCH(t)]
                             [9TRack]
                             [7TRk  ]
                             [7TRack]
```

**Parameters:**

u — specifies the logical unit number by which the tape UDS file will be referenced. It must be a number from 1 to 15 inclusive. No duplicate logical unit number can be defined in one job.

ddn — specifies the step name and data definition name (ddname) by which the tape UDS file will be referenced. Refer to the writeup of the /FILE statement for files for complete details.

TAPE — specifies that the file involved is a tape UDS file.

BLksize(s) — specifies the length of the physical block on tape. It is limited only by the buffer space available. The blocksize should be a multiple of the record size. If not specified, the record size is assumed.

BLocksize(s) — same as BLKSIZE.

LRecl(l) — specifies the size of each record. It must be a value between 20 and 32760. The default record size is 128 bytes. For FORTRAN sequential I/O, the maximum logical record length is 133 (unless the system subroutine BIGBUF is called).

RSIZe(l) — same as LRECL.

VOLume(v) specifies the name of the magnetic tape reel. The name can be from 1 to 6 characters in length and any letter (A-Z) and number (0-9) may be used.

disp specifies the disposition of the tape UDS file. If not specified the system will assume OLD which allows the writing operation on the tape, subject to the write-enable ring on the tape reel. SHR may be specified to allow read operation only.

DENsity(d) specifies the density of the tape. For 7-track magnetic tapes, the valid densities that can be specified are 200, 556 and 800 bits per inch (BPI). For 9-track magnetic tapes, the valid ones are 800, 1600 and 6250 BPI. If a 7-track tape is specified, by specifying 7TRK or 7TRACK, the density will be assumed as 800 BPI if it is not specified. For a 9-track tape (9TRK or 9TRACK), MUSIC will assume a density of 1600 BPI if the density is not specified. You can also specify DEN(LOW) or DEN(HIGH) to use the lowest or highest density available for the tape drive. Note that you do not need to specify the density if you are only reading from the tape (not writing).

9TRack
7TRk
7TRack
9TRk
Specifies whether the tape is to be mounted on a 7 or 9 track tape drive. Your installation may not have both types available for your use. See your installation about the types of tape drives available. MUSIC assumes 9TRK (9TRACK) if the density is not specified or is specified as 800, 1600 or 6250 BPI. 7TRK (7TRACK) will be assumed if the specified density is either 200 or 556 BPI, or if the parameter TRTCH is specified.

TRTCH(t) specifies the tape recording technique. This is valid only for 7-track tapes. The following is a table of a list of tape recording techniques and their corresponding actions.

| TRTCH | PARITY | DATA CONVERSION | TRANSLATION |
|---|---|---|---|
| C | ODD | YES | NO |
| OC | ODD | YES | NO |
| E | EVEN | NO | NO |
| ET | EVEN | NO | YES |
| O | ODD | NO | NO |
| T | ODD | NO | YES |
| OT | ODD | NO | YES |

The *Data Converter* feature of 7 track tape drives causes 4 six-bit characters (24 bits) to be written on the tape for every 3 eight-bit bytes sent from your program. During a read operation, this feature reverses the process causing 4 six-bit tape characters to be converted to 3 eight-bit bytes. This conversion feature is done by the tape unit itself as it reads the tape. Even though the tape can move at the same speed with this feature active, the effective read/write rate is reduced to about 75%.

The *Translate* feature of 7 track tape drives causes eight-bit EBCDIC code bytes (the normal internal processing unit representation), to be written on the tape as six-bit BCD characters. Conversely, 6-bit BCD characters read from the tape are translated into their eight-bit EBCDIC equivalent. This translation feature of the tape unit takes no additional time to perform this operation.

*Notes:*

1. Multiple magnetic tapes can be read by reading each file with the END= specification on the READ statement. Multiple magnetic tapes can be written by the use of ENDFILE followed by further WRITEs.

2. Magnetic tapes must be of record format *fixed* or *fixed-blocked*.

3. Magnetic tapes may be used only by jobs run on batch. (Users may submit batch jobs from their work-stations. See the discussion of the SUBMIT program for further details.)

4. There is another form of /FILE statement which can be used to define tape UDS files. The syntax of this form is:

   ```
   /FILE TAPE=(u,X'mn',BLK=s),RSIZ=r,VOL=v,DISP=d
   ```

   This form is used by the older versions of MUSIC. Users are NOT encouraged to use this form of the /FILE statement as it is not as powerful as the new form and will not be supported at some later time. Refer to the User's Guide of older versions of MUSIC for a description of the parameters used.

**Examples:**

```
/FILE 2 TAPE BLK(800) RSIZ(80) VOL(MYTAPE)
/FILE GO.DATA TAPE LRECL(80) VOL(OLDTAP) SHR
/FILE 11 TAPE RSIZ(80) VOL(ABCDEF) 7TRK TRTCH(OC)

/ID TAPEJOB          UUUU,SSS,003,100,100
/FILE 1 TAPE 9TRK BLK(1600) RSIZ(80) VOL(MYTAPE)
       DIMENSION A(100)
       .
       .
       .
       WRITE(1,10)A
10     FORMAT(10I5)
       STOP
       END
/END
```

## Adding data to the end of a tape file:

It is possible for MUSIC FORTRAN, ASSEMBLER, COBOL and PL/I programs to add data records to the end of an existing tape file. This is similar to the DISP=MOD feature of OS. The procedure is:

a. Read the existing records until end of file (EOF) is reached. Physically, this means that a special tape mark record has been read.

b. Backspace one record. This backspaces over the EOF tape mark record, so that the tape is positioned immediately after the last data block.

c. Write the new records.

With OS-mode programs (COBOL and PL/I), the LEAVE option must be specified when the tape file is closed at the end of step (a), to prevent automatic rewinding of the tape. In PL/I, use ENVIRONMENT(LEAVE) on the CLOSE statement. See the example below.

Step (b) can be done in FORTRAN by the BACKSPACE statement. There is no equivalent statement in COBOL and PL/I, but the MUSIC library subroutine BACKSP can be used. The calling sequence is:

```
CALL BACKSP('ddname  ')
```

where the character string argument is the 8-character ddname of the tape file. If the ddname is shorter than 8 characters, it must be padded on the right with blanks. In PL/I, BACKSP must be declared as EXTERNAL ENTRY OPTIONS (ASSEMBLER).

**Sample PL/I program:**

```
/FILE DD1 TAPE VOL(MYTAPE) LRECL(80) BLK(3200) OLD
/LOAD PLI
SAMPLE: PROC OPTIONS(MAIN);
DCL DD1 FILE RECORD SEQUENTIAL,
    BACKSP EXTERNAL ENTRY OPTIONS(ASSEMBLER INTER),
    REC CHAR(80);
ON ENDFILE(DD1) GO TO EOF;
OPEN FILE(DD1) INPUT;
LOOP: READ FILE(DD1) INTO(REC);
    GO TO LOOP;
EOF: CLOSE FILE(DD1) ENVIRONMENT(LEAVE);
CALL BACKSP('DD1     ');
OPEN FILE(DD1) OUTPUT;
REC='ADDED1'; WRITE FILE(DD1) FROM(REC);
REC='ADDED2'; WRITE FILE(DD1) FROM(REC);
CLOSE FILE(DD1);
END;
```

# /FILE (Miscellaneous)

This category includes unit record I/O devices and files that are pre-allocated to the user once signing on to MUSIC is complete. No other parameters are required except those listed above.

**Syntax:**

```
Miscellaneous

/FILE  u    type
       ddn
```

**Parameters:**

u — specifies the logical unit number by which the unit record I/O device or file will be referenced. It must be a number from 1 to 15 inclusive. No duplicate logical unit number can be defined in one job.

ddn — specifies the step name and data definition name (ddname) by which the unit record I/O device or file will be referenced. Refer to the writeup of the /FILE statement for files for complete details.

type — specifies the type of unit record I/O device or file involved. The valid types that can be specified are the following:

| | |
|---|---|
| RDR | this is the same as the default logical unit number 5 which is used to read data from the input stream following the /DATA statement. The maximum record size is 80 bytes. Refer to *Chapter 4. File System and I/O Interface* for more information. |
| PRT | this is the same as the default logical unit number 6 which is the workstation if the program is being run from a workstation, or the high speed printer, if the program is being run from batch. In addition the PRT parameter can be used to send printed output directly to a system printer. (See following section on /FILE statement for printers). The maximum record size is 133 bytes with the first character as a carriage control character. Refer to *Chapter 4. File System and I/O Interface* for more information. |
| PUN | this is the same as the default logical unit number 7 which is the card punch. This is mainly used if your job is being run from batch. The maximum record size is 80 bytes. Refer to *Chapter 4. File System and I/O Interface* for more information. |
| TERM | this is the same as the default logical unit number 9 which is used to read input conversationally from your workstation. Your job will temporarily pause waiting for you to respond to the read. From batch, a read on TERM is taken as if it were one on RDR. The maximum record size is 80 bytes. |
| HOLD | this is the same as the default logical unit number 10 which is used to temporarily store the program output in the *Holding File* until it can be saved when the job is over. You can save the output in the Holding File by using the SV command when your workstation is next in *Go status. The maximum record size is 80 bytes. Refer to *Chapter 4. File System and I/O Interface* for more details about the Holding File. |
| DUMmy | this specifies a dummy file. A read from this dummy file will cause an end-of-file condition, while the output written to it will be ignored. |
| UNDEFined | this specifies an undefined file. A read or write on this file will cause an error message to be issued. |

**Examples:**

```
/FILE  GO.PRINT  PRT
/FILE   5  TERM
/FILE  15  DUM
```

```
*Go
list sample
*In progress
/FILE   7   RDR
/FILE  12   PRT
10     READ(7,*,END=50)A
       B=SQRT(A)
       WRITE(12,30)A,B
30     FORMAT(10X,F10.3,5X,F10.3)
       GO TO 10
50     STOP
       END
/DATA
78.45
27.96
85.66
 .
 .
 .
 ETC.
```

# /FILE (Printers)

This form of the /FILE statement is used to send program output directly to one of the system printers. The first character of each output record should be a valid carriage control for the printer in question. The maximum record length is 133. The output is placed on the print queue and scheduled for printing when the printer becomes available.

**Syntax:**

```
Printers

/FILE  u    PRT(name) [COPIES(num)] [FORMS(forms)] [SPACE(n)]
       ddn
```

**Parameters:**

| | |
|---|---|
| u | specifies the logical unit number which the program uses to reference the printer. It must be a number from 1 to 15 inclusive. No duplicate logical unit number can be defined in one job. |
| ddn | specifies the step name and data definition name (ddname) which the program uses to reference the printer. Refer to the writeup of the /FILE statement for files for complete details. |
| name | specifies the printer location name of the printer to be used. |
| num | specifies the number of copies required. |
| forms | specifies on which type of forms the output is to be printed. |
| SPACE(n) | *n* specifies the amount space needed. SPACE(50) is the default. |

**Examples:**

```
/FILE  6 PRT(SYSTEM)
/FILE  SYSPRINT PRT(ROOM112) COPIES(10)
```

*Note:* The valid values for the *name* and *forms* fields are dependent on your local MUSIC configuration.

## /ID

This statement is used to identify your batch job. For a full description of this statement refer to the topic "Format of the Batch /ID" in *Chapter 3. Using Batch*.

## /INCLUDE

This statement serves as a pointer to a file in the Save Library. It may be placed in the input file as well as in files. You may use many /INCLUDE statements if you wish in one job. When this statement is encountered during job processing, the named file will be located and read instead of the /INCLUDE line itself. When the named file has been read, the reading of the original file will be resumed.

This /INCLUDE feature allows your program or data to be stored in multiple files. For example, each subroutine could be saved as a separate file. The /INCLUDE facility allows the user complete freedom of how and when to split up the user's program and/or data into files.

The /INCLUDE statement is not processed as such during EDIT and SAVE operations. This feature allows you to create and maintain files which contain /INCLUDE statements. The /INCLUDE statements only take effect when your program is executing.

/INCLUDE statements may be *nested* if desired. That is, an included file may itself contain /INCLUDE statements pointing to files which themselves contain /INCLUDE statements, etc. The maximum level of nesting is 5, with the first include statement pointing to the second level.

Include functions can also be controlled during program execution by use of the following subroutines. Refer to *Chapter 9. System Subroutines* of this publication for further details on these routines.

SYSINE, SYSINL, SYSINM, and SYSINR

If a /INCLUDE statement refers to a file saved with the XO (exec-only) attribute, then the only lines that may precede the /INCLUDE statement are /SYS, /FILE, /COM, /INFO, and /ETC. For batch jobs, /ID and /PAUSE may also precede the statement.

If a /INCLUDE statement refers to a file which has a record length of longer than 80 bytes, then only the first 80 bytes of each record will be processed. If the entire record length of the file has to be processed, then the file must be referred to by a /FILE statement as previously described.

**Syntax:**

```
/INCLUDE    name[([NEST][,EOF][,ERRS][+m-n])]
/INC
```

**Parameters:**

name — the name of a file saved in the Save Library. In order to access another user's file that was saved with the SHR option, you must prefix the file's name with the owner's userid as in the example userid:ABC.

NEST — /INCLUDE statements found within the named file are to be processed. If NEST is not specified, any /INCLUDE statements will be treated as data lines rather than statements. This option is the default up to the point in time that a line other than a /INCLUDE, /SYS, /FILE, /COM, or /ETC is found in the input job stream. From this time on, /INCLUDE statements found in *included* files will be treated as data lines unless the /INCLUDE statement which *includes* the file specifies the nest option.

EOF — causes an end-of-file condition after reading the last line of the named file. If the NEST option is also in effect then the EOF condition occurs at the end of each nested file. If not specified, the EOF condition occurs only after all lines of all files have been read.

ERRS — causes an end-of-file condition for any error associated with the reading of any included file. Errors that cause this condition include FILE NOT ACCESSIBLE, a bad /INCLUDE statement, etc. See system subroutine SYSINE for further information.

+m-n — specifies which lines in the file to include from line *m* to *n.*

*Note:* The options specified must be enclosed in parentheses.

**Examples:**

```
/INCLUDE ABC
/INCLUDE USERID:ABC
/INC ABC(NEST)
/INC ABC(NEST,ERRS)

*Go
list sample
*In progress
/include alan
/include roy
/include wilf(nest)
*End
*Go
sample
*In progress
      (program output)
```

**Messages:**

`/INCLUDE xxxxxx  ERR02 USER OR PROGRAM ERROR`
No name was specified for a dynamic /INCLUDE requested by a program, or a /INCLUDE was incorrect.

`/INCLUDE xxxxxx  ERR04 FILE NOT ACCESSIBLE`
The specified file name was not found in the Save Library, or is a private file belonging to another user, or is an execute-only file.

`ERROR WHILE READING FILE xxxxxx  ERR03`
Either a system error was encountered while reading the file, or the file's contents does not match its

record format. This could happen if an attempt is made to /INCLUDE a load module file which was created with record format FC (the default). Load modules should be created with record format F.

## /INFO

This statement is used to inform SUBMIT about where the job is to be submitted to. If specified, it must be the first statement in the job. The first parameter must be the name of the processor (system) to which the job is to be submitted. If *processor* is specified as *MUSIC*, the job will be submitted to MUSIC batch for processing. Consult your installation for a list of other valid processors. The keyword parameters *(kw1, kw2, ...)* are used to specify values *(value1, value2, ...)* which will be used in the appropriate control statements of the job. These parameters are dependant on the particular processor to which the job is submitted, and are usually used to specify items such as time and page limits, passwords, output destinations, etc.

Refer to the description of "Submitting Jobs to MUSIC Batch and Other Operating Systems" in *Chapter 3. Using Batch* of this guide for full details.

**Syntax:**

```
/INFO        processor  [kw1(value1)]  [kw2(value2)]  . . .
```

## /JOB

This statement is used to specify certain parameters that affect processing of the job. This statement is normally used to pass parameters to the MUSIC loaders or to the Linkage Editor.

One blank must appear between the /JOB and the first parameter. As usual, each parameter is separated from each other by commas. Invalid parameters are sometimes ignored. The parameters which may be used with each processor are described in *Chapter 8. Processors*.

**Syntax:**

```
/JOB         parms...
```

## /LOAD

This statement is used to designate a particular processor to be used to process the lines following. The /LOAD statement usually can be followed by a /OPT statement specifying options for the particular compiler or assembler and by a /JOB statement specifying parameters for the loading step. In all cases the processor loads and executes the object code unless /JOB NOGO has been specified. /LOAD statements following the first one are not allowed.

**Syntax:**

```
/LOAD       name
```

**Parameters:**

name — where name is one of the following:

APL — specifies that the APL subsystem is to be invoked. See the discussion of MUSIC/APL in the subsystems chapter.

ASM — specifies that the lines following form a program written for the VS Assembler (and optionally, object modules), and are to be processed by the Assembler and the resulting object program is to be loaded and executed using the OS/MUSIC interface.

ASMLG — specifies that the lines following consist exclusively of object modules to be loaded and executed using the OS/MUSIC interface. Utilization of processing unit time is optimized if this statement is used instead of /LOAD ASM.

COBLG — specifies that the lines following consist exclusively of object modules to be loaded and executed using the OS/MUSIC interface. Utilization of processing unit time is optimized if this statement is used instead of /LOAD COBOL.

COBOL — specifies that the lines following are an VS COBOL program (and optionally, object modules), and are to be processed by the VS COBOL Program Product compiler and the resulting object program is to be loaded and executed using the OS/MUSIC interface.

COBOL2 — specifies that the lines following are an VS COBOL II program (and optionally, object modules), and are to be processed by the VS COBOL II Program Product compiler and the resulting object program is to be loaded and executed using the OS/MUSIC interface.

C370 — specifies that the lines following are a C program (and optionally, object modules), and are to be processed by the C/370 compiler and the resulting object program is to be loaded and executed using the OS/MUSIC interface.

EXEC — specifies that a load module contained in a file is to be executed. The data set must have been defined on a previous /FILE statement.

FORTG1 — specifies that the lines following are a FORTRAN IV (G1) program (and optionally, object modules), and are to be processed by the IBM FORTRAN (G1) Program Product Compiler and the resulting object program is to be loaded and executed.

GPSS — specifies that the lines following are a GPSS V program, and are to be processed by the General Purpose Simulation System program.

IEFBR — specifies that a program that does nothing except return control to MUSIC. This program is useful when you wish to create or delete a UDS file via a /FILE statement. Refer to the /FILE statement writeup for an example of a usage of this program.

LKED — specifies that the lines following consist of object modules and Linkage Editor control lines to be processed by the Linkage Editor.

LOADER — specifies that the lines following consist exclusively of object modules to be loaded and

executed by the MUSIC loader. Utilization of processing unit time is optimized if this statement is used instead of /LOAD BASIC or FORTG1.

VSPASCAL specifies that the lines following consist of source statements to be processed by the IBM VS/PASCAL compiler. Object modules may also be present.

PLI compile and execute a PL/I program. This processor invokes the PL/I Optimizing Compiler, and then the loader. After loading, the program is automatically executed using the OS/MUSIC interface (unless /JOB NOGO is used or the compiler return code is 12 or more). The /OPT and /JOB control statements may be used with this processor. PL/I object modules can optionally be produced and saved, intermixed with source. The object modules are identical with those produced on OS.

PLILG load and execute a PL/I program in object module form. It is more efficient to use this processor than /LOAD PLI when the input consists solely of object modules. The /JOB control statement may be used with this processor to specify LOADER options.

REXX Specifies that the lines following are to be processed by the high level language REXX (Restructured Extended Executor). REXX allows the use of MUSIC statements directly in the REXX program.

RPG specifies that the lines following are an RPG II program and are to be processed by the RPG II and the resulting object program is to be loaded and executed.

RPGAUTO specifies that the lines following are RPG auto report control statements and are to be processed by the RPG auto report processor and the resulting RPG II program is to be compiled and executed.

RPGLG specifies that the lines following are RPG II object modules and are to be loaded and executed.

VSBASIC Specifies that the lines following are to be processed by the VS BASIC compiler. Refer to *Chapter 8. Processors* of this guide for more information about this compiler.

VSFORT specifies that the lines following are a FORTRAN program (and optionally, object modules), and are to be processed by the IBM VS FORTRAN compiler and the resulting object program is to be loaded and executed.

XMON specifies that a COBOL, VS Assembler or VS Fortran program in load module form is to be executed using the OS/MUSIC interface. The load module file must be defined on a previous /FILE statement. The load module is run using the OS/MUSIC interface.

XMPLI execute a PL/I load module created by the MUSIC Linkage Editor. This is similar to /LOAD XMON. It results in faster loading (compared with /LOAD PLILG) and allows overlay structures to be used. Usage is the same as for /LOAD XMON except that the load module file must not be on I/O unit number 4.

XMRPG executes an RPG II load module created by the MUSIC Linkage Editor.

*Note:* For further information regarding the use of each processor, see the description in *Chapter 8. Processors*.

# /OPT

This statement is used to specify parameters for MUSIC language compilers. If used, it must appear following the /LOAD or /JOB statement. See *Chapter 8. Processors* for parameters which may be used with each language compiler. This statement may be omitted if default parameters are to be used.

**Syntax:**

```
/OPT        parameters
```

# /PARM

This statement is used to pass information to a program. A /PARM is automatically generated by some /EXEC and implied EXEC statements. If several /PARM statements are encountered, only the first one is used.

The character string *parameters* (with leading and trailing blanks removed) can be obtained by the program by calling the PARM subroutine described in *Chapter 9. System Subroutines*.

**Syntax:**

```
/PARM        parameters
```

# /PASSWORD

This statement is used in batch to provide the user's batch password. For more information about batch processing, refer to *Chapter 3. Using Batch*.

**Syntax:**

```
/PASSWORD=xxxxxxxx
```

**Parameters:**

xxxxxxxx is the 1- to 8-character batch password that must be supplied when submitting batch jobs.

# /PAUSE

This statement is used on batch to send messages to the operator. The operator has to respond to this message before your batch job continues. For more information about batch processing, refer to *Chapter 3*.

*Using Batch.*

**Syntax:**

```
/PAUSE message
```

**Parameters:**

message — is the text of your message which is sent along with your batch job. This message might contain information about special handling for your job.

# /SYS

This statement is used to specify a limit to the processing unit time to be used by a job. If the time requested on the /SYS statement is greater than the maximum allowed in the user's User Profile, the latter is used. The /SYS statement must appear before any /LOAD statement in order to be correctly handled. /SYS can also specify other options such as job region size.

**Syntax:**

```
/SYS [NOPRINT][,TIME=nnn][,NOSKIP][,REG=nnn][,NEXT=filenm             ]
             [,TIME=nnS]                     [,NEXT=filenm/parameter  ]
             [,TIME=MAX]                     [,NEXT=!filenm/parameters]

       [,CD=\directory][,DEBUG]
```

**Parameters:**

NOPRINT — specifies that control statements (such as /FILE) are not to be printed for this job. If this parameter is specified it should be the first parameter, with one blank space between /SYS and NOPRINT.

TIME=nnn — maximum job time (in units of 60 service units) for which the job is to be allowed to run. When this time is reached, the job is automatically terminated. If the form nnS is used, *nn* specifies the maximum time in service units. If MAX is specified, the maximum time limit allowed in the User Profile is used. The TIME parameter is ignored for a job run from batch since the batch time limit is given on the /ID statement. This parameter may be omitted if the default time limit specified in the User Profile is desired.

Some users may be limited to a maximum they can specify. This limit may be different for prime and non-prime time. You can find the maximum time limits allocated for your userid by running the PROFILE program and specifying the option PRINT.

NOSKIP — for batch jobs, causes the system to suppress the automatic skip to a new page at the bottom of every page of printed output on the high speed printer.

REG=nnnn — specifies the region size desired, in units of 1K = 1024 bytes. For example, "/SYS REG=512" results in a region size of 512K. If this parameter is not specified, MUSIC will

usually assume a region size of 256K (some processors may assume a larger size region). The region size specified should be an exact multiple of 4. Thus, valid sizes include 108, 112, 116, etc. If multiple /SYS statements are used in the same job, the largest region size specified is used. This allows a user to increase a default region size specified within a program file. To specify a region smaller than given on a later /SYS statement, use a Y after the size, for example /SYS REGION=200Y. The Y causes subsequent REGION options to be ignored.

NEXT — specifies the next program to execute after the current program terminates. When the file name is followed by a slash, the text following the slash will be passed to the next program as a parameter list. To make the next program non-cancellable, place a ! directly before the file name. The NEXT parameter must be the last parameter on the /SYS line.

CD=\directory — specifies the current directory to be used for the job. "directory" is anything that can be used as a parameter on the CD (Change Directory) command. The specified directory is made the current one when the job starts. Examples:

```
/SYS CD=\MYDIR
/SYS CD=\              <-- root dir
```

DEBUG — invokes the DEBUG utility as the first user program in the user region. Userid privileges are required.

**Examples:**

```
*Go
list sample
*In progress
/SYS TIME=8S
    1 GO TO 1
      END
*End
*Go
sample
*In progress
MAIN   = 0000FE
003448 BYTES USED
EXECUTION BEGINS
*VSM006 TIME ESTIMATE OF 8 SUs EXCEEDED. JOB TERMINATED
*End
*Go
```

MUSIC/SP User's Reference Guide

# Chapter 7. Using The Editor

# Chapter 7. Using The Editor

## Overview of the Editor

This chapter describes the Editor component of the MUSIC system and is divided into the following main sections:

- This initial section describes Editor concepts including modes of operation.
- Starting and ending an edit session and description of the EDIT command.
- Editor full-screen mode and program function key support.
- Creating a new file using input mode of the Editor.
- Common editing functions.
- Using the prefix area.
- Editor commands, arranged alphabetically. Included is information on command syntax.
- Advanced features: starting column suffixes, logical commands, defining function keys, hexadecimal input and output, and other topics.
- Editor macro facility in REXX.
- Editor restart facility.

## Editor Concepts

The Editor is a general purpose interactive utility for creating, modifying, viewing, and storing files.

At the beginning of the edit, the Editor copies the specified file to a temporary area. All requested changes during the edit session are made to this temporary copy. The changes are not permanent until a FILE, SAVE, or EXEC command is used, which creates a new original file from the current temporary copy.

The Editor is used to create, look at, or modify files. A file consists of lines of information arranged in sequential order. These lines are called records. The Editor can handle records up to 8192 bytes (characters) long.

The Editor is not subject to the user's execution time limit. This enables a userid which has a small time limit to do lengthy edit operations.

### What Can the Editor Do?

- search for lines based on their contents.
- make global changes throughout the file.

- move and copy text and lines.
- add and delete text and lines.
- replace the original file or create another file.
- merge part or all of another file into the current file, and vice-versa (store part or all of current file into another).
- The Editor can be tailored to suit your own needs by defining your own function keys.
- you can create your own editor commands and macros.
- you can program the Editor to be used with your own programs.
- you can execute MUSIC commands and programs from the Editor.

## Workstation Modes for the Editor

The Editor can display the lines being edited in various ways, depending on the type of workstation used. A workstation can be either a terminal or a personal computer. IBM 3270-type terminals are commonly used for mainframe computers, and personal computers accessing mainframes will often emulate the characteristics of 3270-type terminals. 3270-type workstations can support full-screen applications such as the Editor.

Full-Screen Mode — IBM 3270-type workstations use *full-screen* mode for the Editor. Full-screen mode displays an entire section of the file. It lets you make changes by typing directly over the displayed text, and lets you use function keys for many editing operations. Full-Screen mode is described later in this chapter.

Screen Mode — Another mode for editing is called *screen* mode. It is intended mainly for ASCII (i.e. non-3270) screen workstations that are not emulating 3270s. It displays a section of the file but does not allow direct modification of text on the screen or use of function keys.

Line Mode — If neither full-screen nor screen mode is used, the Editor works with only one line at a time. Hard-copy terminals, those without screens, would used this method.

## Editor Modes

The Editor operates in one of two modes: command mode and input mode. The Editor starts off in command mode unless the file you are editing is empty (then you start in input mode). In command mode, the Editor accepts commands and performs them for you. To change from command to input mode, use F11 or the INPUT command. In INPUT mode, you can type multiple lines without interaction with the Editor.

# Starting and Ending the Editor

This section describes the procedures for invoking the Editor and ending the edit session. Once invoked, the Editor operates in full-screen mode for most workstations. See the section entitled "Editor Full-Screen Mode" for details. Information about typing in data is covered in the section "Creating a New File". For information about making changes to a file, refer to the section "Common Editing Functions".

The diagram below is an overview of the necessary steps for starting and ending an edit session.

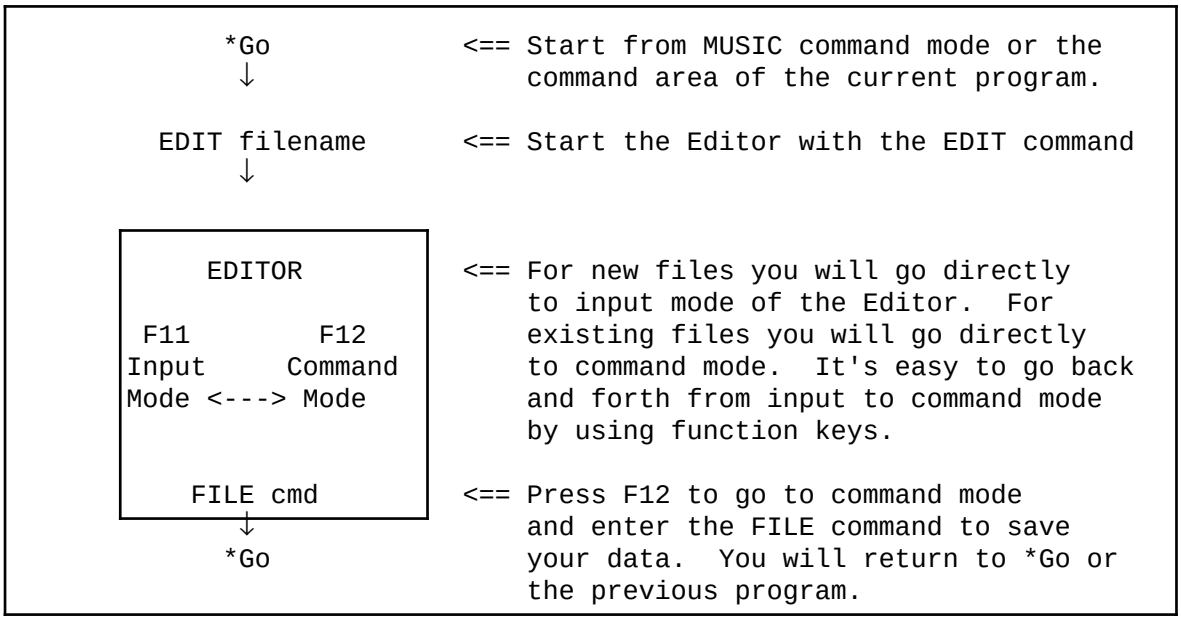



*Figure 7.1 - Steps for Using the Editor*

## Starting the Editor

The Editor can be started in a variety of ways. It can be invoked from *Go, from the command area of a program, or as a selection from a menu facility such as FSI (Full Screen Interface) or TODO (Time, Office, and Documentation Organizer). Also, some MUSIC programs invoke the Editor automatically for collecting data. For example, the MAIL program invokes the Editor when you are ready to type in the text of your message.

There are two MUSIC commands that can be used to start the Editor: EDIT and BROWSE.

EDIT — The EDIT command (abbreviation E) is used most often to start the Editor. The syntax for this command is described below.

BROWSE — Use the BROWSE command (abbreviation B) to edit a file which you do not intend to change. BROWSE is equivalent to the EDIT command with the RO (read-only) option.

**EDIT Command Syntax**

```
EDIT [name  ] [NEW] [LRECL(n)] [F ] [RO] [NOLOG] [UIO] [MAX(n)]
E    [NEW   ] [OLD] [LR(n)   ] [FC]
     [/INPUT]       [nn      ] [V ]
     [/HOLD ]                  [VC]
     [/REC  ]
```

```
Examples:    EDIT MYFILE
             E FILE25 NEW
             E FILE30;L xx
```

**Parameters:**

name — This specifies the name of the file to be edited. If the file name NEW is specified, the Editor goes into Input mode and a completely new file may be created. If the file name /INPUT is specified or the file name is omitted completely, the Input file will be edited. If the file name is omitted, the other parameters must be omitted also. The file name /HOLD can be specified, to edit the output holding file. The special name /REC edits the recording file created by the RECORD command.

Note: to edit a file whose actual name is NEW, specify the userid of the file's owner in front of the name. For example, EDIT ABCD:NEW. The OLD option can also be used, as in EDIT NEW OLD.

NEW
OLD — If NEW is specified, the Editor assumes that the named file is a new file and goes into Input mode automatically. If the named file actually does exist, then when a FILE, SAVE, or EXEC command is issued during the edit session the Editor will ask the user whether the user wants to replace the file or not. If OLD is specified, the Editor will look for the file in your library and go into command mode. OLD is the default.

LRECL(n)
LR(n)
nn — This specifies the record length to be used during the edit. A number from 1 to 8192 can be specified. The record length option should be entered as LRECL(n) or LR(n). If the number *n* contains at least 2 digits (as in 25 or 03), the LRECL keyword can be omitted, and the option entered as simply *n*. The record length will be assigned to the new file when a SAVE, FILE or EXEC command is performed. If not specified, the original record length of the named file (if the file is an existing file), or a record length of 80 (if the file is a new file) will be used. Note that if *n* is less than the record length of the original file, lines may be truncated as they are read by the Editor. For variable length records, *n* represents a maximum record length.

F
FC
V
VC — This specifies the record format to be attributed to the file being edited. This record format will be assigned to the new file when a SAVE, FILE or EXEC command is performed. Either F (fixed format), FC (fixed compressed), V (variable length), or VC (variable compressed) can be specified. Refer to *Chapter 4. File System and I/O Interface* for a description of the record formats. If not specified, the original record format of the named file (if the file is old), or FC (if the file is new) will be used.

RO — Causes the Editor to use Read Only mode. This is equivalent to the BROWSE MUSIC command. In full-screen mode, the text on the screen is protected (non-modifiable). RO implies the NOLOG option and the Editor command "SUBSET 2". If you decide you want to make changes, use the "SUBSET 0" Editor command to revert to normal editing.

NOLOG — Suppresses the restart feature of the Editor. This prevents the Editor from looking for a log file or creating a new one. Refer to the discussion on the restart facility later in this chapter.

UIO — Causes the Editor to read the file by 512-byte blocks, using MFIO UIO requests. Each block becomes a record for the edit. This lets you edit (but not change) record format U files, or files that cannot be read sequentially. The record length for the edit is automatically set to 512. UIO implies RO and NOLOG. Do not use the FILE or SAVE command to write to the file being edited, since UIO is not used for the writes.

MAX(n) Specifies the maximum number of records the Editor is to read from the file being edited. This option implies RO and NOLOG. It is useful for looking at the first part of a large file. Example: EDIT BIGFILE MAX(500).

The parameters can be specified after the file name on the EDIT command in any order. A blank or a comma must be used between parameters.

Editor commands can also be specified at the end of the EDIT command. A semicolon (;) is used to separate the commands. For example, EDIT SAMPLE;L XY;C/XY/123/ instructs the Editor to edit the file named SAMPLE, locate the first line which has the character string XY in it, and then change the string XY to 123.

## Ending the Editor

The function key F3 (Quit) is used in the Editor to exit the Editor program **without** saving any changes. The following commands are needed to save your changes and update the file stored in your library.

```
FILE [newname]
```

When this command is entered, the Editor saves any changes done during the current edit session, and ends the session.

The original file is updated, unless you changed the file name during the edit session (see NAME command later). Another way to leave the original file alone is to specify a different file name on the FILE command.

Examples:

```
1.  file                    2.  file program2
```

Example 1 saves the file with the name displayed at the top of the Editor screen. Example 2 uses another name and the name displayed at the top of the screen is ignored.

```
SAVE [newname]
SA
```

Same as FILE except the edit session continues.

```
EXECUTE [newname]
EX
```

Same as the FILE command above except the file is executed also.

# Editor Full-Screen Mode

## Introduction to Full-Screen Mode

Users of 3270-type workstations take advantage of the special features of these workstations in *full-screen* mode of operation (default). (For information about other modes of operation see the topic "Workstation Modes for the Editor" earlier in this chapter.)

In full-screen mode, you can modify data directly on the screen, using the local editing keys, as well as enter commands in an area near the bottom of the screen. Also, various editing operations can be done by pressing program function (PF or F) keys to enter Editor commands.

## Screen Format for Full-Screen Mode

The screen display for full-screen editing is divided into the following areas:

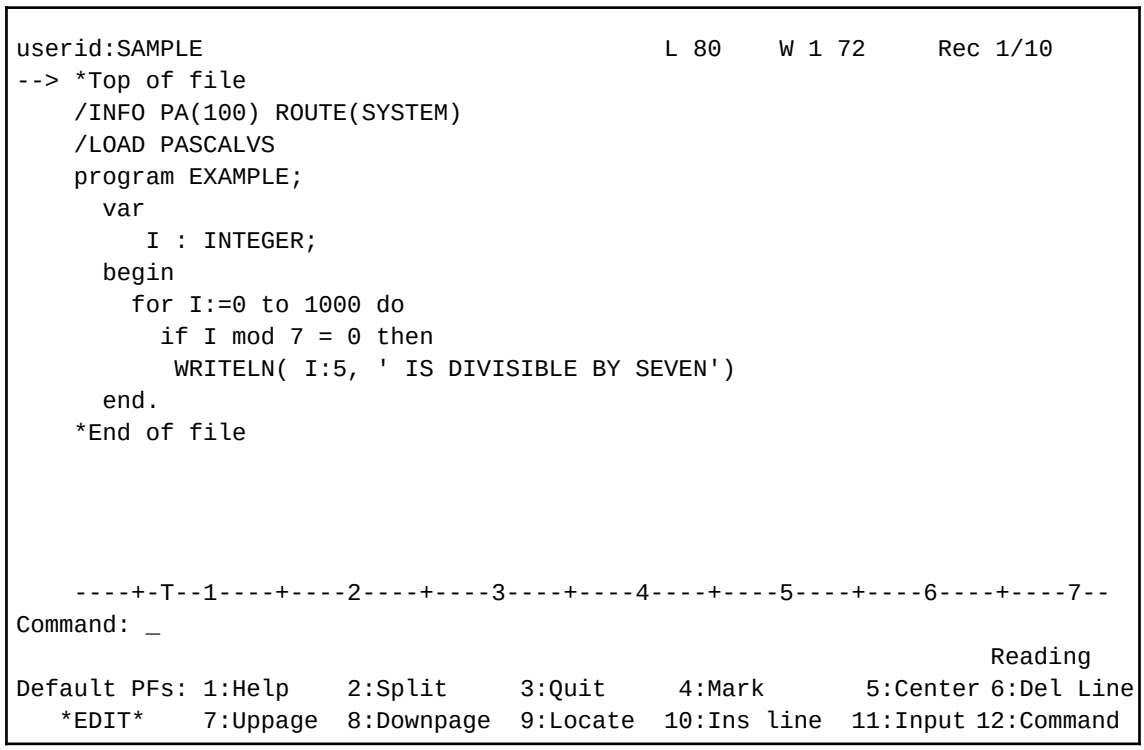



*Figure 7.2 - Screen Display in Full-Screen Mode*

1. **Title line:** this is the first line on the screen, and contains the name of the file being edited, the file's logical record length (L), the starting and ending window columns (W), and the current line number with the total number of lines (Rec) in the file. This field is not modifiable on the screen, although Editor commands can be used to change the file name and window setting.

2. **Lines of the file:** this is the main body of the screen, immediately following the title line. It may be up to 20 lines on a 24-line screen, or 39 lines on a 43-line screen. Each screen line displays one record of

the file, and the text is directly modifiable on the screen. Only the window portion of each record (see the WINDOW command) is displayed, to a maximum of 72 characters (124 on wide-screen terminals). The current line is indicated by an arrow pointer in the left margin and is also displayed in high intensity (red on a color terminal). If line numbering is in effect (the NUMBER command), line numbers are displayed in the left margin. If the PREFIX command is in effect, then a 4-character modifiable area indicated by "====" is in the left margin. The beginning of the file, if displayed, is indicated by `*Top of file.` Similarly the end of the file is indicated by `*End of file.` In Input mode, several empty lines are displayed following the current line, thus allowing you to add new lines to the file by typing them on the screen.

3. **Tab line:** this shows column numbers and input tab positions. The tab positions (displayed as T's) correspond to the column numbers specified on the TABIN command. The tab positions are relative to the starting window column, rather than to column 1 of the file's records. See the topic "Defining a TAB Key" later in this chapter.

4. **Command area:** this is used for entering Editor commands. Several commands may be entered by separating them by the command delimiter character, normally semicolon (;). If the Editor is in input mode rather than command mode, the command area is replaced by the centered message `* Input Mode *.`

   *Note:* REXX Editor macros and MUSIC commands can also be entered in the command area.

5. **Message area:** this line of the screen is reserved for messages from the Editor. Up to 3 messages can be packed into this area. The messages are displayed in high intensity (white on a color terminal). If the messages do not fit, or if considerable output is generated by a command such as SCAN, the output is displayed on a new screen in normal MUSIC format, with the status message `More...` in the bottom right corner. Pressing the ENTER key allows you to continue. Editor commands may not be entered until the `More...` condition has been cleared. While output is being displayed in normal MUSIC format, the PA1 key may be used to go to attention mode (**Attn**) and skip output (by the /SKIP command).

6. **Status indicator:** the current status of the workstation and the Editor is shown in the bottom right-hand corner of the screen. Possible status words are `Reading, Working, **Attn**,` and `More...`

7. **SHOW area:** if a "SHOW filename" command has been used, lines of the specified file are displayed at the very bottom of the screen. By default, the area is used to display the definitions of the function keys. Refer to the SHOW command for more information.

## General Notes

1. Since at most 72 characters are displayed from each line of the file, the largest possible window width (as set by the WINDOW command) is 72. (For a wide-screen terminal such as the 3270 model 5, the maximum width is 124 rather than 72.) When full-screen mode is initially turned on and when the WINDOW command is used, the horizontal window setting is automatically adjusted if necessary. Also, the zone (refer to the ZONE command) is modified at this time to match the window setting. The resulting zone is from column 1 to the end of the window. This zone may be changed by a subsequent ZONE command.

2. If a line contains unprintable or nonstandard characters, avoid using direct screen changes to modify the line. This is because any change to the line causes unprintable characters in the line to be replaced by blanks. Use the CHANGE command instead.

3. A logical input tab character may be used when typing lines on the screen. The tab characters are expanded to the appropriate numbers of blanks when the next action key is pressed. When the command TEDIT is used (rather than EDIT), or when the Editor command TEXT SCRIPT is used, input tab characters are not expanded.

# Editing Keys

The Editor uses a full screen technique to let you change data on the screen and manipulate the file by using 3270 *local editing* keys and *action* keys in addition to Editor commands.

## Local Editing Keys

The workstation stores in its own memory the lines displayed on the screen. Changes can be made to these lines, without interacting with the host computer, by using local editing keys. For example, the DELETE and INSERT keys are useful for removing or adding characters to a field. The ERASE EOF key deletes the characters from the cursor position to the end of the field (in this case to the end of the screen line or command area).

Characters can be replaced by first positioning the cursor at their location and then typing over them. The cursor position is changed by using the various arrow keys (cursor control keys) on the keyboard. The NEW LINE key (with a *down and to the left* arrow on it) and TAB key are useful for moving the cursor to the start of the next line. This key and the other arrow keys repeat when held down. For NET3270, the NEW LINE key is Ctrl-Enter, or the * on the right-hand keypad.

Changes made with local editing keys are only transferred to the Editor's temporary copy of the file when an action key (a function key or the ENTER key) is pressed (see below).

If you inadvertently make unwanted changes when using local editing keys, the CLEAR key cancels the effect of all screen changes made since the last action key was pressed. The ERASE INPUT key (a local key), which clears all unprotected fields, should not be used.

*Note:* If the BROWSE command was used to start the edit, or the RO (read-only) option was used on the EDIT or TEDIT command, then the file's text can not be changed on the screen. If you try to type over text on the screen, the workstation will go into the *input inhibited* condition. Press the RESET key to clear this condition.

## Action Keys

The keys which cause an interaction with the system are referred to as *action keys*. They are ENTER, the program function (PF or F) keys F1 to F24, CLEAR, PA1, PA2, and SYS RQ (or TEST REQ). The following describes the default definitions of the action keys.

ENTER — This key transmits screen changes, input lines, and commands to the system. It can also be used to advance to the next screen when `More...` appears in the bottom right corner. If no other keys were pressed since the last action key, ENTER moves the cursor to the current line.

PA1 — Not normally used. Refer to the description of the screen message area (above) and the S parameter on the SCAN and CHANGE commands (below) for more information.

PA2 — This key is used for multi-session control. It adds or deletes a MUSIC/SP session, or switches to the previous or next session.

*Note:* Make sure you have pressed ENTER or one of the function keys before pressing PA2. Otherwise any screen changes you have entered since the last action key are lost. If you press PA2 by mistake and do not wish to go to another session, press the ENTER key.

CLEAR Cancels the effect of any screen changes made since the last action key was pressed, redisplays the current screen, and places the cursor in the command area. Input mode is terminated if it was in effect.

SYS RQ
TEST REQ This key is normally not used. It has the same effect as the CLEAR key, except that the work-station is placed into MUSIC attention mode. You may then enter an attention mode command (such as /TIME) or a blank line. The Editor eventually redisplays the original screen (you may need to press the ENTER key). Use of the TEST REQ key may be prohibited or restricted by the system environment.

**Function Keys (F1 - F12) for the Editor**

Function keys have been defined to perform certain EDITOR commands, eliminating the need to type them. They provide a fast and easy way to edit files. Function keys can be used both in command and input modes of the Editor.

It is possible for your private EDITOR file or the system *COM:EDITOR file to change the definitions of the function keys. Enter the command SHOW PF to see the actual function key definitions in effect for your edit session. If you want to change the default definition of function keys, see the topics "Defining Function Keys and the X command" and "Creating your own Editor" later in this chapter.

F1, F13 (HELP) provides information about how to use the Editor. You are presented with a list of topics and are asked to enter the number(s) of the topic(s) you want more information about.

F2, F14 (SPLIT) splits a line into two lines. The character at which the cursor is positioned becomes the first character of the second line.

In Browse: (PRINT) Prints the file. Pressing F2 places "PRINT *CUR" into the command area. You can fill in a printer name if you wish, by adding " r(name)" onto the end of the PRINT command. Then press ENTER to complete the print operation, or press CLEAR to cancel it.

F3, F15 (QUIT) terminates the Editor session without performing any save operation. If you have made changes to the file but have not issued a SAVE command to make the changes permanent, you will be prompted for permission to end the edit session. Enter YES or Y to end the session, or NO or N to cancel the QUIT operation and continue editing.

F4, F16 (MARK) is used to designate (mark) a line or group of lines. To mark a group of lines, mark the first and last line of the group. The marked group can be used by commands such as MOVE., COPY., DELETE., and STORE. Marked lines are displayed with a vertical bar character in the left margin of the screen.

In Browse: (TOP) Goes to the top of the file.

F5, F17 (CENTER) causes the screen to be redisplayed so that the current line (or the one containing the cursor when the function key is pressed) appears in the middle of the screen display. The screen display is shifted the appropriate number of lines up or down in the file in order to center the current line. The UPWINDOW and DOWNWINDOW commands perform a similar function.

F6, F18 (DELETE) removes the current line from the file. Note that if the cursor points to a line on the screen when the function key is pressed, that line is the one which is deleted. This is because the current line pointer is moved to the cursor before the function key request is done. The same applies to the other function keys. To delete a line, position the cursor on it and press F6. This function key should not be used if prefix area commands are entered on the same screen.

In Browse: (LAST) Goes to the bottom of the file.

F7, F19 (UPPAGE) displays the previous page (screen) in the file (towards the beginning of the file).

F8, F20 (DOWNPAGE) displays the next page (screen) in the file (towards the end of the file).

F9, F21 (LOCATE) locates the next occurrence of the character string used on the last LOCATE, SEARCH, FIND, HUNT, UPLOCATE, or UPFIND command. The search starts at the line following the current line. If the CURSOR LOCATE command is in effect, the cursor will be positioned at the start of the found string when the screen is displayed.

F10, F22 (INSERT) inserts a blank (null) line after the current line. This is useful for adding a single line to the file. You can type the line over the inserted blanks.

In Browse: (WINDOW LEFT) Shifts the display window up to 72 columns to the left.

F11, F23 (INPUT FLIP) puts the Editor into Input mode. This is useful for entering several lines after the current line. In Input mode, the screen displays the current line and a few lines above it, followed by null lines in the remainder of the screen. The user types over these lines in order to enter the new lines. The NEW LINE local key is used to go to the next line. If more space is needed, press the ENTER key. If the Editor is already in input mode, F11 terminates Input mode. In this way, F11 *flips* back and forth between Input and command mode.

In Browse: (WINDOW RIGHT) Shifts the display window up to 72 columns to the right.

F12, F24 (CMDPFK) moves the cursor to the command area. This is necessary for entering Editor commands that are not provided for by function keys. The arrow keys may also be used to place the cursor in the command area prior to entering commands. F12 also terminates Input mode. The Editor command corresponding to this function key operation is CMDPFK.

*Notes:*

1. Other action keys should not be used.

2. The ERASE INPUT key should not be used. It cancels all screen changes made since the last action key was pressed and erases the screen display. If you hit the ERASE INPUT key by mistake, press CLEAR to redisplay the screen.

3. After pressing an action key, the INPUT INHIBITED indicator is on briefly while the system processes the request. On 3277 terminals, INPUT INHIBITED is indicated by a bright square at the right of the screen. On newer models of terminals, it is indicated by a large X at the bottom of the screen. If INPUT INHIBITED comes on before an action key is pressed, it means that you have tried to modify a protected part of the screen; press the RESET key to continue.

## Order of Operations

It is possible to combine several operations at the workstation before pressing an action key. Operations are done by the system in the following order:

1. Changes made to text on the screen.
2. Movement of the current line pointer to the line indicated by the cursor, if no prefix area commands are entered.
3. Prefix area commands.
4. Command-area commands.
5. Program function key operation.

Note that the cursor must be placed into the command area before typing an Editor command. The cursor can be positioned by using any of the arrow keys. It can also be put into the command area by using the F12 key.

If the Editor is in Input mode when a function key is used (standard or user-defined), Input mode is terminated before the function key request is done.

## Suggestions for Efficient use of Full-Screen Mode

- If your 3270-type workstation has more than 12 function keys, set up a private EDITOR file to define extra function key operations in addition to (or in place of) the standard ones. Using function keys is faster and easier than typing commands. Good candidates for function key operations are the commands: MOVE., COPY., DELETE, DELETE., WINDOW FLIP, JOIN, TOP, BOTTOM, DUP, UPLOCATE, MINSERT, MDELETE, UPWINDOW, DOWNWINDOW, FILE, ECHO, CURSOR END, CURSOR n TEMP, NULLS FLIP, NUMBER FLIP, SUBMIT.

- Don't use the ENTER key in input mode to end each line. Use the NEW LINE key or TAB key. The NEW LINE key has a *down and to the left* arrow on it, and is located on the right-hand side of the alphabetic keyboard. This gives much better response in Input mode, since no system interaction is required (NEW LINE and TAB are local keys). Once the screen is full or nearly full with input lines, press the ENTER key to get a fresh screen. Outside of Input mode, the NEW LINE or TAB key is often quicker than the command function key (F12) for positioning the cursor in the command area. For NET3270, the NEW LINE key is Ctrl-Enter, or the * on the right-hand keypad.

- Use the INPUT function key (F11) or the command function key (F12) to terminate Input mode, rather than pressing ENTER twice. This requires only one interaction with the system. The input function key (F11) places the cursor at the last input line; the command function key (F12) places the cursor in the command area.

- When you know that you will be typing a command in the command area (rather than making a change on the screen), end the previous action by using the command function key (F12) instead of the ENTER key. This ensures that the cursor will be positioned in the command area, and may eliminate one interaction with the system.

- Several Editor commands can be entered in the command area by separating them by a semicolon (;). This technique can really speed up editing, but use it carefully, since an error on one of the commands does not stop execution of the remaining commands.

- To retrieve the previous command line entered in the command area, place the cursor at the tab line before pressing an action key. This redisplays the last non-blank text entered in the command area. You may then modify and re-issue the command, or the command may be re-issued (as is) by simply pressing the ENTER key.

- The screen cursor may be used to move the current line pointer. Simply use the arrow keys to place the cursor at the desired new current line. That line becomes the new current line before any command area commands or function key operations are done. Of course, a command or function key operation may further move the pointer. Positioning the cursor at the `*Top of file` line is equivalent to a TOP command. Positioning it at the `*End of file` line is equivalent to a BOTTOM command.

# Creating a New File

The first step to creating a new file is to invoke the Editor. If you are choosing an edit selection from one MUSIC's menu facilities (like FSI) then you will be prompted for the necessary information. Otherwise, you need to use the MUSIC command "EDIT" with a file name and the NEW parameter. For example,

```
EDIT myfile new
```

starts the Editor with an empty file named "MYFILE". (For full details about the EDIT command, see the earlier topic in this chapter called "Starting the Editor".)

The figure below shows the Editor's screen display for input mode. The Editor starts automatically in input mode for new files.

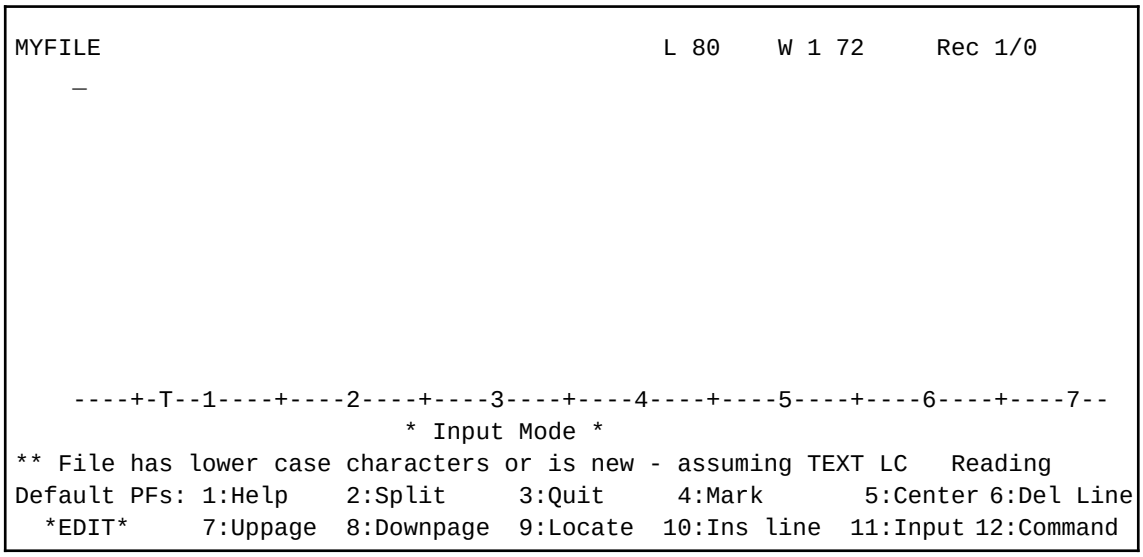



*Figure 7.3 - Screen Display for Input Mode*

In the figure above your cursor is positioned on the first input line on the screen. You are ready to type in your data. Remember to use the TAB key to end each line. When you are finished and want to save the file, press F12 to go to the command area, type FILE, and press ENTER.

## Input of Data for the Editor

There are five important rules to know for typing in data. Take the time now to understand and learn the correct procedures.

1. You can correct typing mistakes on the line you are currently typing by using the BACKSPACE key or <-- key. Just backspace over the incorrect characters and retype.

2. When in input mode, use the TAB key to skip to the next line, rather than the ENTER key. The TAB key shows an arrow pointing to a vertical line (-->|). (The TAB key is a local key and does not interact with the system.) Once the screen is full or nearly full with input lines, press the ENTER key to get a fresh screen.

*Note:* You do not have to wait until you get to the bottom of the screen before pressing ENTER. However, if you press ENTER after typing only a single line, the Editor will automatically bring you to command mode. You will need to press F11 to return to input mode.

3. It is important to stay inside the boundaries (in the input area) on the screen. You can go out of bounds by moving the cursor with some of the arrow keys (-->, <--, etc.). If your cursor is accidentally moved out of the input area and you try to type, the keyboard becomes LOCKED. Press the RESET key to unlock the keyboard, and use the TAB or arrow keys to move the cursor back to the input area.

4. When adding blank spaces between words, use the SPACEBAR and not the right arrow key (-->). Arrow keys are for moving the cursor around the screen. In the example below, the line was typed using the incorrect --> key for spaces. Notice the results after the ENTER key is pressed.

| Before | After |
|---|---|
| `10  FORMAT(' THE ROOT IS')` | `10FORMAT('THEROOTIS')` |

5. When the Editor is invoked, a message appears at the bottom of the screen indicating whether the command TEXT LC or TEXT UC is in effect. TEXT LC means that lower case letters are retained and are not automatically converted to upper case (default for new files or existing files already containing lower case). TEXT UC means that all letters are automatically converted to upper case when an action key is pressed (default for existing files that contain only upper case letters.) To change the default setting, issue the appropriate TEXT command.

# Common Editing Functions

This topic covers the most common editing functions. When you need to do more complicated editing, like moving text, or making global changes, refer to the section in this chapter entitled "Editor Commands".

Local editing keys and function keys are used to perform most editing tasks, the exception being the FILE command, which is used to end your edit session. The following describes the most common editing tasks.

## Making Changes on the Current Screen

Make changes to the text displayed on the current screen by moving the cursor to that position and typing over the existing text. Some areas of the screen, such as the title line, are protected and cannot be changed. If you attempt to type in one of these areas, the keyboard will lock. Press RESET to unlock the keyboard, so you can move the cursor to a valid input area.

## Moving the Cursor

There are several arrow keys for moving the cursor on the screen.

| ↑<br>up | ↓<br>down | ←<br>left | →<br>right | →\|<br>tab | \|←<br>tab |
|---|---|---|---|---|---|

Be careful not to move out of the input area with the UP, DOWN, LEFT, and RIGHT arrow keys. Use the TAB key to skip to the next line. (Remember to use the SPACEBAR when you want to add blanks and text to the end of a line. The arrow keys only appear to leave spaces. As soon as an action key is pressed, the blanks disappear.)

## Paging Up and Down

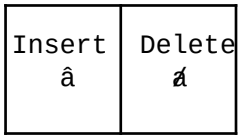
To see the previous screen use the F7 key (UPPAGE). To see the next screen use the F8 key (DOWNPAGE). When you need to see part of the previous or next screen, position your cursor on a line and press F5 to center that line on the screen.

| F7<br>uppage | F8<br>downpg | F5<br>center |
|---|---|---|

Commands such as TOP, BOTTOM, and LOCATE can also be used to move up and down in your file. They are described later in the topic "Editor Commands".

# Inserting and Deleting Characters

Two important keys for editing are the INSERT and DELETE keys. They are local editing keys and should not be confused with the function keys F10 key (INSERT LINE) and F6 key (DELETE LINE).



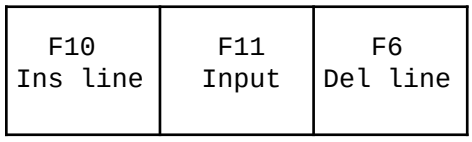

Insert Key — Position the cursor where you want to insert new characters and press the INSERT key. Type in the desired character(s). The new characters are placed in front of the old text. If you fill up the entire line the keyboard will lock. If you need more room, position the cursor where you want to end the line and press F2 to split the line in two.

Either the word INSERT or a symbol appears at the bottom of the screen indicating that INSERT mode is in effect. To cancel INSERT mode, press the RESET key or the INSERT key a second time, depending on your workstation.

Delete Key — Position the cursor on the characters to be deleted. Press the DELETE key to remove each unwanted character.

# Inserting and Deleting Lines

There are two function keys for inserting lines and one function key for deleting lines. To use these keys, position the cursor on a line and press the appropriate key. If the cursor is in the command area, the line pointer (-->) indicates where to insert or delete.




Ins Line — Position the cursor on a line and press F10 to insert 1 blank line after the line the cursor is on.

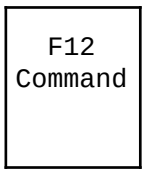
Input — Position the cursor on a line and press F11 to go to input mode. When you have several lines to add, input mode is more efficient than pressing the F10 (Ins line) key for each new line.

Press F12 (Command) to return to command mode. (F11 works as a toggle key, it also returns to command mode, but the cursor is left on the current line.)

Del line — Use F6 to delete the line the cursor is on.

## FILE and QUIT Commands

When you are ready to save your file and exit the Editor, do the following:

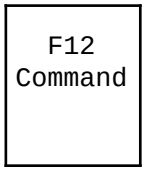



**- press F12 to go to the command area**
**- type the command: FILE**
**- press ENTER**

If you do not want to save any changes, press F3 (Quit).

## Getting Help for the Editor

Press F1 to get help for the Editor. A list of topics is displayed for you to choose from. For help on a particular command, do the following:




**- press F12 to go to the command area**
**- type the command: HELP with a command**
**- press ENTER**

## Common Editor Commands

Editor commands are issued in the command area on the screen. In command mode you can move into the command area using the TAB key or F12 key. In input mode, press F12 to go to the command area.

Commands can be typed in upper or lower case letters. After entering a command, you must press ENTER to have the Editor process it. An example of an Editor command is "FILE".

*Note:* MUSIC commands are also allowed from the Editor's command area. Use a slash (/) to distinguish MUSIC commands from Editor commands. You can use the EDIT command to edit another file while remaining in the first edit session. More information about MUSIC commands is in *Chapter 5 - MUSIC Commands*.

The following Editor commands are grouped into task-similar categories. Here is a table of the most common commands:

| Moving in the File | Changing the File | Working with Marked Groups | |
|---|---|---|---|
| FIND | CHANGE | MARK | UNMARK |
| LOCATE | MERGE | COPY. | MOVE. |
| TOP | | DELETE. | CHANGE. |
| BOTTOM | | STORE | |

| Ending the Edit Session | Miscellaneous |
|---|---|
| FILE | HELP |
| SAVE | DUP |
| EXECUTE | NAME |
| QUIT (F3) | PURGE |

There are many Editor commands available for novice and advanced users. For complete documentation on all Editor commands, refer to the section in this chapter entitled "Editor Commands" or or type "HELP COMMANDS" in the Editor's command area.

# Marking a Group of Lines

When editing a file, the need often arises to define a set of consecutive lines within the file, and then perform some operation on those lines. Examples of this are moving or copying a group of lines from one part of the file to another, restricting a global change to a section of the file, converting a set of lines to upper case, and deleting a group of lines.

The MARK command and the *dot* forms of commands such as MOVE, COPY and CHANGE, provide this capability in the MUSIC Editor. Use of marked groups greatly reduces the need for line numbers while editing.

To define a group of lines, the user moves the line pointer to the first line of the group and issues the MARK command, then moves the pointer to the last line of the group and issues the MARK command again. Once the group has been defined, it can be referred to in various commands by using a period (.) as a parameter. For example, the command CHANGE/ABC/DEF/.G changes every occurrence of ABC to DEF within the group. The commands MOVE. and COPY. move and copy the group. These and other "." commands are described below.

Although the MARK command and the various . commands can be used on any type of workstation, they are best suited to full-screen mode on a 3270-type workstation. In full-screen mode, each line of a marked group is identified by a vertical bar in the left margin on the screen. The user's private EDITOR file would define a program function key as MARK, and probably two other keys as MOVE. and COPY..

### Some Notes on the MARK Command

Only one set of consecutive lines can be marked at a time. The marked group is maintained until a new group is defined, or until an UNMARK command is issued, or until all the lines of the group are deleted. As new lines are added to or deleted from the file, the marks are adjusted accordingly. The command MARK ? reports which lines are currently marked. The blank between MARK and ? may be omitted. Abbreviations for the MARK and UNMARK commands are MA and UNMA.

When defining a group, the first and last lines may be marked in either order. The group may consist of only

one line, in which case the second MARK command is not needed. Once a multi-line group has been marked, or a 1-line group has been marked and used in a . command, a subsequent MARK command starts a new group.

**Commands for Operating on a Marked Group**

In these commands, the . character is a command parameter, and the blank between it and the command name may be omitted. Also, the command name may be abbreviated, as in MO., CO., and DEL..

= . Moves the current line pointer to the first line of the group. This is handy for returning to a particular point in the file. The command is usually entered as =. to save typing.

MOVE . Moves the marked group of lines to after the current line. The group remains marked. For example, if lines 51 through 75 are marked and the current line is 20, then MOVE. is equivalent to MOVE 20 51 75

COPY . Copies the marked group of lines to after the current line. The original group remains marked.

DELETE . Deletes the marked group. Note that the MDELETE command is another (sometimes faster) way of deleting a block of lines. The operation may be qualified by a logical expression as the second parameter, as in DEL.,(NOT/ABC/), which deletes all lines of the group which do not contain ABC.

CHANGE A period (.) can be used in place of the *n* (line count) parameter on a CHANGE or logical CHANGE (CHANGEL) command. This causes the change to apply to each line of the marked group (subject to the logical expression, if CHANGEL is used). The Editor automatically goes to the first line of the group before starting the change. Example: C/ABC/DEF/.F

REPEAT . Causes the next BLANK or OVERLAY command to apply to each line of the marked group. It also moves the current pointer to the first line of the group, and places the screen cursor into the command area.

TOUC .
TOLC . Converts the marked group to upper or lower case.

COPYCOL n1 n2 n3 .
When . is used as the 4th parameter, instead of a line count, the COPYCOL operation is applied to each line of the marked group.

SEQ LINES=. When the LINES= parameter specifies a period rather than a line count, the sequencing operation is applied to each line of the marked group. Example: SEQ 1 1 COL=15 LEN=3 LINES=.

SORT . This Editor macro uses the MUSIC SORT command to sort the marked group.

STORE name Writes the marked lines to the specified file. "A" for append can be added after the file name to have the text added to the end of an existing file (i.e. STORE abc a).

PRINT . Prints the marked group.

# Prefix Area

The prefix area, turned on by the command PREFIX ON, is a 4-character modifiable field at the left of each displayed line of the file. It contains ====, or the last 4 digits of the line number if NUM ON is in effect. You can enter various special commands in the prefix area, to do editing operations such as inserting or deleting lines, moving and copying lines, etc.

The following diagram illustrates the screen display after the PREFIX ON command is entered:

```
userid:SAMPLE                                L 80    W 1 72     Rec 1/10
> ==== *Top of file
  ====  /INFO PA(100) ROUTE(SYSTEM)
  ====   /LOAD PASCALVS
  ====  program EXAMPLE;
  ====  var
  ====       I : INTEGER;
  ====    begin
  ====      for I:=0 to 1000 do
  ====        if I mod 7 = 0 then
  ====         WRITELN( I:5, ' IS DIVISIBLE BY SEVEN')
  ====    end.
  ====  *End of file



    ----+-T--1----+----2----+----3----+----4----+----5----+----6----+----7--
Command: _
                                                                    Reading
Default PFs: 1:Help    2:Split     3:Quit     4:Mark       5:Center 6:Del Line
  *EDIT*     7:Uppage  8:Downpage  9:Locate  10:Ins line  11:Input 12:Command
```

*Figure 7.4 - Prefix Area of the Editor*

The prefix area is controlled by the commands:

```
PREFIX ON                              NOPREFIX
PRE    OFF
       FLip
       CLear
```

**Parameters:**

ON is assumed if there is no parameter

OFF turns off prefix area

FLIP reverses the previous setting

CLEAR clears any pending prefix operations

The following prefix commands are automatically defined. Command names can be entered in upper or lower case. "n" is an optional number, which usually defaults to 1 if omitted.

| | |
|---|---|
| In | Inserts *n* lines after this line. |
| An | (Add) Same as In. Inserts *n* lines. |
| Dn | Deletes *n* lines, starting at this line. |
| D* | Deletes to end of file. |
| DD | Indicates first and last line of a group of lines to be deleted. |
| Cn | Copies *n* lines. |
| CC | Indicates first and last line of a group of lines to be copied. |
| Mn | Moves *n* lines. |
| MM | Indicates first and last line of a group of lines to be moved. |
| F | (Following) indicates a target for an operation such as move or copy. The lines are moved/copied to after this line. |
| P | (Preceding) indicates a target for an operation such as move or copy. The lines are moved/copied to before this line. |
| "n | Duplicates the line *n* times. |
| ""n | Duplicates a group of lines *n* times. |
| .xxx | Assigns label xxx to the line (as in the POINT command). |
| . | Does a MARK command for this line. |
| <n | Shifts the line's text *n* columns to the left. |
| >n | Shifts the line's text *n* columns to the right. |
| <<n | Shifts a group of lines left *n* columns. |
| >>n | Shifts a group of lines right *n* columns. |
| INP | Starts Input Mode after this line. |
| POW | Starts Power Input mode after this line. See the command POWERINP. |
| /n | Makes this line the current line and places the cursor in the nth position (column *n)* of the record. If column *n* is outside the current window display, the window is shifted left or right as needed. If *n* is not specified, the window is not changed, the cursor is placed at the beginning of the window, and the specified line becomes the new current line. This operation is done after other prefix operations. |
| /E | Makes this line the current line and places the cursor after the last nonblank in the text field. This operation is done after other prefix operations. |
| /END | Same as /E |
| CE | Centers the display around this line. |
| J | Joins the next line to the end of this line (JOIN command). |
| Ln | Converts *n* lines to lower case. |
| Un | Converts *n* lines to upper case. |
| LL | Converts a group of lines to lower case. |
| UU | Converts a group of lines to upper case. |
| FC | (Flip Case) Reverses the case of each letter in the line. |
| FF | Formats a group of lines, using Script. The left and right margin columns can be specified by the MARGINS macro: "margins left right" The default margins are 1 65. Type FF in the prefix area of the first and last lines of the group of lines to be formatted. Other associated macros: FF, UNFF, and FORMAT. |
| WW | Formats a group of lines, without using Script. It formats the text by invoking the Rexx macro $FLIN. All 3 parameters of the MARGINS command are honoured. The default MARGINS values are 1 65 0. Type WW in the prefix area of the first and last lines of the group of lines to be formatted. A blank line causes a formatting break, and can be used to separate paragraphs. Unlike FF, Script control words like .SP are not honoured. WW is faster than FF for small sections of text. WARNING: There is no undo for WW. |

The optional numeric parameter n can be omitted, in which case the value 1 is used. n can be entered before

or after the command name, e.g. nI or In. For paired commands (such as <<), n can be entered on either the first or second command. If entered on both, the parameter entered on the line nearer the end of the file is used. Similarly, if mXn is entered, n is used and m is ignored.

Care must be taken when line numbering is in effect and numeric values are entered in the prefix area, since the "background" digits of the line number become part of the entered text, and are indistinguishable from the digits actually typed. The editor processes the text by ignoring any leading or trailing digits that are the same as the original "background" digits. For example, if the line number is 1234 and you type i13 over "123", then the editor sees the entered text as i134, ignores the trailing 34, and the prefix command is processed as i1. If i13 is meant, you should type i13b (b=blank).

Some prefix operations do their work by MARKing a group of lines, so they UNMARK any previously marked lines. These are D (in FLAG mode only), DD (in flag mode only), "", <, >, <<, >>, L, U, LL, UU, and FF.

Prefix operations are done in the order they were entered, except that operations entered on the same screen (by the same 3270 action key) are done from top to bottom of the screen. An exception to this rule is that the / prefix command (/n and /E) are done last. Also, an operation that requires more than one entry (such as DD or CC) is not done until it has been completely specified. Until then, it is "pending", and a pending message appears on the tab line.

Input Mode is turned off before a prefix operation is done. This means that the INP prefix command has no effect when combined with other prefix operations, unless it is done last.

Each prefix operation is done by an editor macro, so REXX ON (default) is required. You can define your own prefix operations, by using the DEFINE PREFIX command and writing the appropriate macro. See the DEFINE command description for more information.

# Editor Commands

## Notation

The notation *string* indicates a sequence of blanks, letters, etc. Leading blanks in character strings are considered part of the string starting with the second position after the command name. In the command descriptions, square brackets are used to indicate optional items.

## Command Syntax

Parameters of commands may be separated by one or more commas or blanks. A blank is not required between the command name and the parameter unless the first character of the parameter is a letter (A to Z). A blank is not required following a flip character or column number suffix. The following commands require a blank in any case: SAVE, FILE, EXEC, RUN, LIST, MERGE, PURGE, INPUT, NAME, STORE. Commands may not be preceded by a slash (/).

In general, any abbreviation between the shortest abbreviation and the full command name may be used. For example, the following may be used for the CHANGE command: C, CH, CHA, CHAN, CHANG, CHANGE. An exception is the PRINT command, which allows P but not PR (PR is the PROMPT command). Another exception is the INSERT command, which allows I and INS but not IN, to avoid confusion with the INPUT command. In the following command descriptions, the shortest abbreviations are shown under the full form of the command.

The command delimiter character (normally semi-colon), the flip character, and all string delimiters must be special characters. They may not be letters, digits or blanks.

In logical expressions, the string modifiers F and Cn (where n is a column number) may be separated by blanks or commas, or the separator character may be omitted entirely. For example, (/ABC/(FC10)). The same applies to options on the CHANGE and SPLIT commands. Examples: C/ABC/DEF/*GV, SPLIT/XXX/C5IC10.

## String Separator

Commands such as CHANGE require that strings be separated by a string separator character. The usual character used is a slash (/), but any non-alphanumeric character except the command delimiter character may be used. The string separator character is self-defining. For example, the first nonblank character after the command CHANGE is automatically defined as the string separator for that command.

## Functional Summary of Commands

The following Editor commands are grouped into several categories. More information about any of these commands can be found after this summary under the heading "Editor Command Descriptions", or by typing "HELP xxx", where xxx is the command name. Some of the commands below are actually Editor macros and the REXX ON command (default setting for the Editor) must be in effect to use them.

## Moving in the File:

| | |
|---|---|
| =n | move to line number n |
| BOTTOM | move after the last line of the file |
| FIND | same as LOCATE, except the string must start in column 1 |
| HUNT | same as FIND, except start at the top of the file |
| LAST | move to the last line of the file |
| LOCATE | locate the next line containing a specified character string |
| LOCATEL | LOC the next line that tests true against logical expression |
| NEXT | move towards the end of the file |
| SEARCH | same as LOCATE, except start at the top of the file |
| SEARCHL | same as LOCATEL, except start at the top of the file |
| TOP | move to the first line of the file |
| UFIND | (Upward FIND) same as ULOCATE, but string in column 1 |
| ULOCATE | (Upward LOCATE) move towards the top, looking for a string |
| UP | move towards the beginning of the file |
| XF | (macro) extended find |
| XL | (macro) extended locate |
| XUF | (macro) extended up find |
| XUL | (macro) extended up locate |

## Displaying lines:

| | |
|---|---|
| CENTER | center the current line in the middle of the screen |
| DOWNPAGE | shift the screen display 1 screen towards the end of file |
| DOWNWINDOW | shift the screen display towards the end of the file |
| LEFT | shift the window display left |
| LIST | display the file being edited, or an external file |
| NONUMBER | turn line numbering off |
| NUMBER | turn line numbering on |
| PRINT | display a number of lines, starting with the current line |
| RIGHT | shift the window display right |
| SCAN | display all lines containing a specified character string |
| UPPAGE | shift the screen display 1 screen towards the top of file |
| UPWINDOW | shift the screen display towards the beginning of the file |
| WINDOW | define starting and ending columns to be displayed |

## Making changes to the file (see also: Formatting):

| | |
|---|---|
| ADD | add text to the end of a line |
| BLANK | set specified characters of a line to blanks |
| CHANGE | make a change to a line or group of lines |
| CHANGEL | make a logical change to a line or group of lines |
| COPY | copy lines from one place to another within the file |
| COPYCOL | copy text from one part of a line (or lines) to another part |
| DELETE | delete a line or group of lines from the file |
| DELETEL | logically delete a line or group of lines from the file |
| DUP | duplicate the current line a no. of times |
| INPUT | begin INPUT mode, to add lines to the file |
| INSERT | add a new line following the current line |
| JOIN | join the next line to the end of the current line |
| MDELETE | delete MINSERT unused lines or a group of lines |
| MERGE | bring a copy of an external file into the file being edited |
| MINSERT | add a group of new lines following the current line |

| | |
|---|---|
| MOVE | move lines from one place to another within the file |
| OVERLAY | overlay part of a line with new characters |
| POWERINP | begin Power Input mode |
| REPEAT | specify the no. of times to repeat the next BLANK or OVERLAY |
| REPLACE | replace an entire line |
| SHIFT | (macro) shift marked lines left or right |
| SORT | (macro) sort a file or part of a file |
| SPELL | (macro) invokes SPELL program |
| SPLIT | break the current line into two lines |
| TOLC | change characters to lower case |
| TOUC | change characters to upper case |

## Getting information:

| | |
|---|---|
| = | get the line number of the current line |
| ATTRIB | get information (attributes) about a file |
| ECHO | cause information to be displayed in the command area |
| GETV | (macro) displays the value of a SETV name |
| HELP | request information about the editor or an editor command |
| NAME | display or set the file name for the edit session |
| NOSHOW | remove text displayed by SHOW command |
| SHOW | display function key or X cmd definitions; from a file |
| SIZE | get the total number of lines in the file |
| SPACE | get library space information for your userid |
| TAG | display or set the tag string of the file being edited |
| TIME | display time of day, date, cpu time, and number of users |
| USERS | same as TIME |

## Ending the edit:

| | |
|---|---|
| END | same as QUIT |
| EXECUTE | store the file (as in FILE command) and then execute it |
| FILE | store the changed file in place of the original file |
| OFF | same as QUIT, but also terminates the MUSIC session |
| QQUIT | (quick quit) terminates without any messages or prompting |
| QUIT | terminate the edit without saving the file |
| RUN | same as EXECUTE, except use the MUSIC input file (/INPUT) |
| SAVE | similar to FILE, but do not terminate the edit |
| SETRC | set a job return code (exit code) for the edit |

## Full-Screen Mode:

| | |
|---|---|
| AUTOSKIP | cause automatic cursor skip at end of text fields |
| CMDPFK | cause the cursor to be put into the command area |
| COLOR | define the color of various parts of the screen |
| CURSOR | control the placement of the cursor on the screen |
| DEFINE | define a program function key or the X command |
| FILL | put blanks at the end of each screen field |
| FS | start full-screen mode (for 3270-type workstations) |
| NOFILL | put nulls at the end of each screen field (the default) |
| NOFS | end full-screen mode |
| NUMBER | turn line numbering on or off |
| PREFIX | turn on or off the modifiable prefix area |

| | |
|---|---|
| WINDOW | control which columns of the records are displayed |

## Formatting:

| | |
|---|---|
| FF | (macro) format marked lines or entire file using SCRIPT |
| FORMAT | (macro) format the file using SCRIPT |
| MARGINS | (macro) adjust margins before using FF |
| UNFF | (macro) restore file to original form before FF was used |
| UNFORMAT | (macro) restore file to original form before FORMAT was used |

## Working with Marked Lines:

| | |
|---|---|
| COPY. | copy marked lines to another place in the file |
| DELETE. | delete marked lines |
| FF . | (macro) format marked lines using SCRIPT |
| MARK | define a group of lines to be operated on |
| MOVE. | move marked lines from one place to another |
| SHIFT | (macro) shifts marked lines left or right |
| SORT . | (macro) sorts marked lines |
| STORE | store marked lines in another file |
| TOLC. | change marked lines to lower case |
| TOUC. | change marked lines to upper case |
| UNMARK | remove the marking for a group of lines |
| =. | go to a marked group in the file |

## Other commands:

| | |
|---|---|
| AIN | terminate hexadecimal input mode |
| ALPHA | terminate hexadecimal output mode |
| ARROW | use arrow pointer form of screen mode display |
| ASMFIX | align assembler source statements to specified columns |
| BEEP | beep the speaker the next time the screen is displayed |
| BOTH | obtain output in both hexadecimal and character form |
| BR | (macro) uses show area to display a file |
| BRIEF | stop automatic verification (displaying) of file changes |
| CALC | (macro) Perform a calculation, e.g. "calc 1+2" |
| CASE | ignore or respect case differences during string searches |
| CD | changes directories |
| CMDS | enable or disable MUSIC (*Go) commands |
| CREP | enable or disable command name replacement |
| DBCS | turn the double-byte character set on or off |
| DEFINE | define a program function key or the X command |
| DELCHAR | define the delete char. to be used by MDELETE & MINSERT cmd. |
| DELIM | change or remove the command separator character (normally ;) |
| ENQ | (macro) prevents more than 1 simultaneous edit of a file |
| FLAG | begin automatic flagging of changed lines |
| FLIP | define a "flip" character, used to control verification |
| HEX | obtain output in hexadecimal form |
| KEYS | (macro) Display and/or change function key definitions |
| LANGUAGE | turns on the language of your choice |
| LOG | control frequency of writing to log file (restart feature) |
| MD | makes a new directory |
| MSG | display a message line at the workstation |

| | |
|---|---|
| MSGS | suppress or enable all messages |
| NOARROW | turn off the arrow pointer form of screen mode display |
| NOCHANGE | indicate that there are no unsaved changes |
| NOFLAG | stop automatic flagging of changed lines |
| NOSCREEN | turn screen mode off |
| NOTRAN | turn output character translation off |
| OKREPL | suppress FILE/SAVE verification prompt for a file name |
| POINT | assign a 1-8 character label to a line |
| PRINT | print a file on a specified line printer |
| PROMPT | define or remove the prompting character |
| PURGE | remove (delete) a file (alias: ERASE) |
| RENAME | change the name of a file |
| REXX | enable or disable the use of Rexx procedures as Editor cmds |
| SCREEN | turn screen mode on |
| SEQ | put sequence numbers into each line of the file |
| SET | (macro) compatible command with CMS editor |
| SETV | assign a character string yyy to a name xxx |
| SUBMIT | submit a job to MUSIC batch or other batch processors |
| SUBSET | define which subset of editor commands is to be allowed |
| TABIN | specify input tab positions |
| TABOUT | specify output tab positions |
| TEXT | specify handling of input lower case and tab characters |
| TRAN | turn output character translation on |
| TREE | (macro) shows graphical display of directories |
| UNDELETE | restore a deleted line |
| VERIFY | control verification (displaying) of changes to the file |
| X | execute a predefined string of commands (see DEFINE) |
| XIN | begin hexadecimal input mode |
| ZONE | define the ending column for some commands such as CHANGE |
| * | signify a comment line |

# Editor Command Descriptions

```
ADD  string
A
```

This command adds *string* after the last non-blank character of the current line.

**Spacing:** One blank character must occur between the keyword and the first character of *string.* Any additional blanks are considered part of *string.*

**Example:** Before: **SAMPLE LINE**
Command: **ADD 1234**
After: **SAMPLE LINE1234**

```
AIN
```

This command can be used to restore alphabetic input format after an XIN command has been used.

```
ALPHA
ALP
ALFA
ALF
```

This command can be used to restore alphabetic output format after a HEX or BOTH command has been used.

```
ARROW
AR
```

(This command is not needed with full-screen editing.) The ARROW command can be used to get a slightly different display format for the old *screen* mode, i.e. the mode set by the SCREEN command. When ARROW is used, the current line is indicated by an arrow in the left margin, rather than by "=====" lines above and below the current line. This is useful on low speed ASCII screens, since the number of characters transmitted is less. Also, the number of lines displayed can be decreased by specifying a number on the SCREEN command. For example: SCREEN 9;ARROW.

```
ASMFIX  [n1,n2,n3]
```

This command is useful when editing an assembler source file. If no parameters are used on this command then it will attempt to align the operation code field to column 10, the parameter field to column 16, and the comments field to column 35. The 3 optional parameters on this command can be used to give other column numbers. All lines of the file are aligned, except assembler comments, object module statements, and lines starting with / or =.

**Pointer:** Set to the top of the file.

```
ATTRIB  filename
AT
```

The ATTRIB command displays some of the attributes of a specified file, namely its logical record length (LRECL), record format (RECFM), number of lines (or 0 if not known), and the file's allocated size in K (1K = 1024 bytes).

| | | |
|---|---|---|
| Record formats are: | F | Fixed-length records, uncompressed. |
| | FC | Fixed-length compressed records. |
| | V | Variable-length records, uncompressed. |
| | VC | Variable-length compressed records. |
| | U | Undefined record format. |

*Note:* The number of lines is especially useful if you wish to merge some lines from the end of the file into the file being edited.

```
AUTOSKIP  ON
AUTOSK    OFF
          INPUT
          NOINPUT
          COMMAND
          NOCOMMAND
```

The AUTOSKIP command controls whether fields on the 3270 screen end with an automatic skip to the next modifiable field. Auto skip is OFF by default, but can be requested for displayed lines of the file: for command-mode lines (COMMAND or C) or Input Mode null lines (INPUT or I) or both (ON). The NOCOMMAND or NOC option turns off auto skip for command-mode lines. The NOINPUT or NOI option turns off auto skip for Input Mode null lines. The OFF option turns off both. The last displayed line of the file never ends with an auto skip. For example, AUTOSKIP INPUT is used by the POWERINPUT command to enable continuous typing in Input Mode.

If no parameter is specified, ON is assumed.

If more than one parameter is specified on the command, each is processed in sequence, as if given on separate commands. For example, "AUTOSK OFF I" is equivalent to "AUTOSK I NOC".

"AUTOSKIP COMMAND" should not normally be used when a PREFIX area is displayed, since typing would continue into the prefix area of the next line.

When NET3270 is used for 3270 emulation, Document Mode (Alt-E toggle) must be off in order for auto skip to work.

**BEEP**

This command causes the speaker at the workstation to beep the next time the FS screen is displayed. It is mainly used for programming purposes.

**BLANK string**
**BL**

This command replaces characters in the current line with blanks. The replacement is position dependent. Characters that appear in the command string replace characters in the same position on the line with blanks. Any character can be used to indicate where you want blanks. *String* may contain up to 78 characters, each of which will cause one blank to appear in the current line. This command is the complement of the OVERLAY command.

The BLANK command is useful in combination with the REPEAT command. If there is a preceding REPEAT command, the BLANK is repeated for the specified number of lines, beginning with the current line.

**Spacing:** One blank should be present between the keyword and *string*. All other blanks are part of *string*.

**Example:**

| | | | |
|---|---|---|---|
| Before: | **SAMPLE LINE** | | |
| Command: | **BL XX** | **or** | **BLC2 XX** |
| After: | **S PLE LINE** | | |

Specifying "BLC2 XX" would have the same results as "BL XX" above. For more information about using a starting column suffix, see the topic "Advanced Features" later in this chapter.

**BOTH**

This command causes lines to be displayed such that each printable character is shown above the left hand portion of the hexadecimal representation. It also temporarily turns off screen mode if it is on at the time the command is issued. The command HEX or ALPHA can be used to tell the Editor to use another output

representation.

```
BOTTOM
B
```

The BOTTOM command moves the pointer to one line after the last line in the file.

**Example:**

```
Before:   SAMPLE LINE          <--Pointer
          ANOTHER LINE
          THE LAST LINE
Command:  B
After:    SAMPLE LINE
          ANOTHER LINE
          THE LAST LINE
                               <--Pointer
```

```
BR filename
```

The BR command uses the SHOW area (the bottom part of the screen) to browse a file. This lets you see 2 files at the same time.

Once browse has started, some extra function keys are defined, as indicated in the display. For example, F19 goes down 1 page in the browsed file, and F15 ends the browse. (On most PC's, you get F(n+12) by pressing Shift-Fn. For example, F15 is Shift-F3.)

Also, while in BR browse, you can enter commands that apply to the browsed file:

```
BR TOP                    BR BOTTOM
BR UP n                   BR DOWN n
BR LEFT n                 BR RIGHT n
BR LOCATE string          BR FIND string
BR =n
```

The normal abbreviations can be used, e.g. BR T, BR B

```
BRIEF
```

This command puts the Editor into BRIEF status, which bypasses display of the new current line after the following commands: ADD, BLANK, CHANGE, CHANGEL, DELETE, DELETEL, FIND, HUNT, LAST, LOCATE, LOCATEL, NEXT, OVERLAY, SEARCH, SEARCHL, SPLIT, UP. Display of the current line does not resume until a VERIFY command is given. (VERIFY is assumed when the Editor begins.)

```
CALC valid-REXX-expression
```

This Editor macro allows you to evaluate a REXX expression, and have the result displayed in the message area. This is useful for performing arithmetic calculations without exiting the Editor or starting another session; sometimes you want the result and the current screen to be displayed together. For example, CALC 11+7 displays the following in the message area:

```
** 11+7 = 18
```

*Note:* Note that a "valid REXX expression" can consist of a simple arithmetic calculation, or a number of REXX function calls.

**Examples:**

```
calc 5*7 + 22/7
calc 2**16              (2 to the power 16)
calc (3.5-1.9)*1.09
calc d2x(26)            (convert decimal to hex)
calc x2d(1a)            (convert hex to decimal)
calc date()             (current date)
calc userid()
```

```
CASE [IGNORE   ]
CA   [I        ]
     [RESPECT  ]
     [R        ]
     [MIXED    ]
     [M        ]
     [UPPERCASE]
     [U        ]
```

The CASE command specifies whether or not differences between upper and lower case characters should be ignored when the editor is searching for a string of characters (IGNORE/RESPECT), and/or whether text entered by the user should be left as is (MIXED) or converted to upper case (UPPERCASE).

The IGNORE/RESPECT setting affects all commands which involve string searches, such as LOCATE, LOCATEL, SCAN, CHANGE, etc. It does not affect changes typed over screen text in full-screen mode, or *string2* of the CHANGE command, or commands such as ADD or INSERT.

With CASE RESPECT, upper and lower case characters are considered different when searching.

With CASE IGNORE (the initial default), upper and lower case characters are considered the same, for the purposes of searching for a specified string of characters. This is as if all characters were in upper case. The file *COM:EDITOR normally contains the command CASE IGNORE, so that setting is the default for most edits.

When CASE is specified without a parameter, the current IGNORE/RESECT setting is shown. To show the

current UPPER/MIXED setting, use the TEXT command without any parameter.

The command CASE MIXED is equivalent to the command TEXT LC. CASE UPPERCASE is equivalent to the command TEXT UC.

More than one parameter can be used on a CASE command. For example, CASE UPPERCASE RESPECT. This can be abbreviated to CA U R.

**Example:**

```
CA I  The command "LOCATE abc" finds "ABC" and "aBc" as well as "abc". The
      command "CHANGE/mcgill/McGill/*" changes "Mcgill", "MCGILL", "MCGill",
      etc. to "McGill" (assuming TEXT LC is in effect).
```

**CD dirname**
**CD..**
**CD**

This command changes directories for tree-structured file system. Same as the MUSIC command CD.

**CENTER**
**CE**

The CENTER command moves the full-screen display up or down in the file, so that the current line will be in the center of the screen display. By default, F5 is defined as CENTER. To shift the display up or down in the file, place the cursor at a line and press F5 to center that line on the screen.

**CHANGE [/string1/string2/][n][G][D][F][Cn][V][P][S]**
**C**

The CHANGE command replaces the first occurrence of a character string with another in a number of lines (normally 1), beginning with the current line. Within the range of lines, only the first occurrence of the string in each line is replaced. The number of lines searched depends on the parameters specified in the command. The last line searched becomes the current line. If the line pointer is at EOF (end of file), an automatic TOP command is done before the change.

When a CHANGE command is entered with no parameters, the editor's CHANGE Panel is displayed. The CHANGE Panel lets you do a change operation by filling in fields on the panel.

The search for the string is affected by whether or not CASE IGNORE is in effect. CASE IGNORE ignores differences between upper and lower case characters. Refer to the CASE command.

The parameters n, G, D, F, Cn, V, P, S are not required. If used, they can be specified in any order. Commas are not required to separate the parameters.

**Parameters:**

string1 Character string which is to be replaced.
string2 Replacing character string. If omitted, *string1* is deleted. The final string delimiter (/) may be omitted if no parameters are used.
n Number of lines searched. If not specified, 1 is assumed. If * is specified, the search will be applied to the rest of the file, starting with the current line.
G If specified, the change will be applied to ALL occurrences of *string1* in as many lines as specified by the *n* parameter.
D If specified, the remainder of the line (or zone) following the changed string is replaced by blanks. ("D" stands for "delete".)
F If specified, the change is made only if "string1" begins in column 1 of the line (or in column n if Cn is used). ("F" stands for "first" or "FIND mode".)
Cn If specified, the change is made only if "string1" occurs starting in column n or later.
V If specified, all lines which are changed are displayed regardless of whether the editor is in the BRIEF status or the VERIFY status. If V is not specified and the range is more than one line, the changed lines are not displayed. ("V" stands for "verify".)
P If specified, the editor prompts for the user's permission to make a change. The user can respond Y (for yes), N (for no), S (to stop the execution of the CHANGE command), G (to continue command execution without further prompts), or = (to find the current line number).
S If specified, it causes the screen to be displayed for each line found (as in 3270 "screen mode"). On a 3270 terminal, the PA2 key must be pressed between screens.

**Examples:**

| | |
|---|---|
| `C/ABC/DEFG/` | Changes the first occurrence of ABC to DEFG in the current line. The line pointer is not changed. |
| `C/OLD/` | Removes the characters OLD from the current line. |
| `C/OLD//3,G` | Removes all occurrences of OLD from the next 3 lines, beginning with the current line. The line pointer is moved down 2 lines. |
| `CH /XXX/YY/ 20,F,C11` | In the next 20 lines, changes XXX to YY wherever XXX occurs starting in column 11. |
| `C/ /*/G` | Changes every blank to * in the current line. |
| `C//AAA/` | Adds AAA to the beginning of the line. |
| `CHANGE/E/ES/*GV` | This will change all occurrences of E to ES in the rest of the file, starting with the current line. Changed lines are displayed. |
| `CHANGE$/$.$` | This changes "/" to "." in the current line. The dollar sign ($) acts as the string delimiter in this case. |

**CHANGEL (logical expression)/string1/string2/[parameters]**
**CL**

This command is similar to the CHANGE command except that, for each line searched, the change will only occur if the logical expression is true for that line.

The *n*, G, D, F, Cn, V and P parameters are also permitted on this command as they are on the CHANGE command.

**Example:**

If it is desired to change all occurrences of the string WRONG to the string RIGHT in all lines which also contain the string MAYBE, but not if the line contains one and only one of the strings HOT and CHA, the commands would be:

```
TOP
CL(/MAYBE/AND.NOT/(/HOT/XOR/CHA/))/WRONG/RIGHT/*GV
```

**CMDPFK**
**CMDPF**

This command is used only in 3270 full-screen mode, and is intended primarily for use in function (PF) key definitions. It causes the cursor to be put into the command area the next time the screen is displayed. The standard definition of F12/24 is equivalent to a CMDPFK command.

**CMDS ON**
**OFF**
**NONE**

The CMDS command controls execution of MUSIC (*Go) commands and programs entered as commands in the editor. Normally, when command xxx is entered in the Editor and xxx is not an Editor command or abbreviation, the system searches for macro xxx (normally file xxx.MAC). If no macro is found, xxx is executed as a MUSIC command (*Go command or the name of a program to be executed).

CMDS ON is the initial setting (no restrictions).

With CMDS OFF, a / must be entered before the MUSIC command or program name. This does not affect execution of macros. CMDS OFF could be used to prevent command typing errors from executing files or *Go commands by mistake.

With CMDS NONE, no MUSIC commands or programs can be executed from within the editor, even with a / (slash).

*Note:* REXX OFF does not prevent execution of explicit MUSIC commands (/xxx), provided CMDS NONE is not in effect.

**CNTINFO**

The CNTINFO macro shows the file usage and creation date of the current file. You can reset a file's count to 0 and set the creation date to today's date. Type "HELP COUNT" for more information.

```
COLOR
COLOR fieldname=color fieldname=color...
COLOR BASE
COLOR DEFAULTS
```

The COLOR (COLOUR) command redefines the color of various parts of the editor screen. This assumes that the workstation supports 3270 extended data streams (i.e. the Start Field Extended order). Not all workstations support all the possible colors. See the topic "Changing Screen Colors" in the section "Advanced Features" later in this chapter for details.

```
COPY  i,j,k
CO
COPY.
CO.
```

The COPY command copies a group of lines from one part of the file to another by reference to the line numbers of the lines. The original group of lines will not be deleted. The last line of the group of lines inserted will become the new current line. The = command is useful for finding the line number of a line.

If the MARK command has been used to define a group of lines, the "COPY." command can be used to copy the lines to after the current line (or to the top of the file if the last command was TOP). For more information about marking a group of lines for copying, see the topic "Marking a Group of Lines" earlier in this chapter.

COPY differs from MOVE in that the MOVE command deletes the source lines, while the COPY command does not.

**Parameters:**

i Line number of the line after which a group of lines is to be copied. If i is 0, the lines are copied to the beginning of the file.
j Line number of the first line of the group of lines to be copied.
k Line number of the last line of the group of lines to be copied.

Line numbers may be specified in any of the following forms: n, *+n, *-n, LAST, LAST-n, where n is a number, * means the current line, and LAST means the last line of the file.

**Examples:**

```
COPY 10 25 30          copies lines 25 to 30 to after line 10.
COPY 42 35 35          copies line 35 to after line 42.
COPY 0 18 20           copies lines 18 to 20 to the start of the file.
COPY * LAST-2 LAST     copies the last 3 lines of the file to after the current line.
```

```
COPYCOL  c1 c2 c3 [n] [NOPROMPT]
COPYC             [.] [N       ]
CC
```

The COPYCOL command copies text from one part of a line to another part, in each of one or more lines starting with the current line. The copied text replaces existing text in the destination field of each line. The last line modified becomes the new current line. This command is not affected by the zone setting.

Parameter *c1* is the starting column number of the text to be copied. *c2* is the ending column number of the text to be copied. *c3* is the starting column number of the destination field. *n* is the number of lines to be modified (the default is 1 line). A dot (.) specifies a marked group of lines. If NOPROMPT or N is specified as a final parameter, the normal verification prompt for this command is bypassed.

**Examples:**

```
COPYCOL 11 15 21 10
```

Copies columns 11 through 15 to columns 21 through 25, in 10 lines starting with the current line. The original contents of columns 21-25 is replaced.

```
CC 40 59 7
```

Copies columns 40-59 to 7-26 in the current line.

```
CC 10 10 20 100 N
```

Copies column 10 to column 20 in 100 lines. No prompt is done.

```
CREP ON
     OFF
```

The CREP command is used to enable or suppress automatic command name replacement. Command name replacement is requested by the DEFINE command in the form:

```
DEFINE CMD xxx yyy
```

The above command would cause the command name XXX (entered by the user) to be processed as YYY. CREP OFF disables command replacement. The definitions are still retained, and can be later re-enabled by CREP ON.

```
CURSOR LOCATE
CU     NOLOCATE
       n
       n TEMP
       END
       PREFIX
       PREFIX TEMP
```

The CURSOR command controls output cursor positioning during full-screen mode.

When the Editor displays the screen, it normally puts the cursor at the first position of the current line or the command area. However, commands such as CURSOR, LOCATE and SPLIT can cause the cursor to be displayed at a different position in the current line.

**Parameters:**

LOCATE — places the cursor at the start of the found string after any of the commands: LOCATE, UPLOCATE, FIND, UPFIND, SEARCH, HUNT. You can put a CURSOR LOCATE command into your private EDITOR file to make this option the default for all your edits. Abbreviation is LOC.

NOLOCATE — cancels the LOCATE option. Abbreviation is NOLOC.

n — is a number from 1 to 72. It is a column position number relative to the start of the screen field for the current line. 1 refers to the first character of the field, 2 to the second character, etc. *n* places the cursor at the specified column position whenever the cursor is not put out in the command area, provided a command such as LOCATE or SPLIT or CURSOR END or CURSOR n TEMP does not result in a different placement. This stays in effect for the remainder of the edit. To cancel the effect of this command, use CURSOR 1.

n TEMP — places the cursor at position *n* in the screen field for the current line, but only for the next screen display. Abbreviation T may be used for TEMP, as in CU 15 T. This command is intended mainly for function key definitions. For example, DEFINE F1 INSERT ABC---XYZ;CURSOR 4 TEMP.

END — is similar to CURSOR n TEMP, except that the cursor is placed after the last nonblank character in the field.

PREFIX — Causes the editor to place the cursor at the first column of the prefix area, rather than at the beginning of the record. This persists until reset by the command CURSOR 1 or CURSOR n. PREFIX may be abbreviated PRE.

PREFIX TEMP — Places the cursor at the first column of the prefix area, but for the next screen display only. TEMP may be abbreviated T, as in CURSOR PRE T.

**Examples:**

```
CURSOR 41
CU 15 T
CUR LOC
```

```
DBCS ON
     OFF
```

The DBCS command is used to turn Double-Byte Character Set mode on or off for the Editor. DBCS mode is needed to handle the special 2-byte characters (representing pictographic symbols) used in some Asian languages such as Japanese (Kanji), Chinese, and Korean.

*Notes:*

1. In order to set DBCS mode on, you must be using a workstation that supports DBCS display and input.

2. When the Editor starts, DBCS mode is automatically set on if a double-byte national language (such as Kanji) is in effect. Also, DBCS mode is set when the Editor LANGUAGE command is used to set a DBCS language, e.g. LANGUAGE KANJI implies DBCS ON. Setting a non-DBCS language turns off DBCS mode, e.g. LANGUAGE ENGLISH implies DBCS OFF.

3. Using the Editor DBCS command affects only the current edit session, and does not change the language setting.

4. Internally, a string of 0 or more double-byte characters is preceded by a Shift-Out (hex 0E) byte and followed by a Shift-In (hex 0F) byte. Normal text (single-byte) and double-byte text can be mixed in the same record.

5. Since case (upper or lower) is not defined for double-byte characters, case is always significant within double-byte text, and double-byte text is not affected by the commands TOUC, TOLC.

6. Character strings in commands such as LOCATE and CHANGE can be double-byte or mixed. However, the command name and syntax items such as delimiters and options must be single-byte text.

7. The Editor SPLIT command in DBCS mode operates on the entire record, rather than on just the ZONE part of the record.

See also: LANGUAGE.

```
DEFINE  Fn     commands
DEF     PFn    <TAB>
        PFPn   <RETRIEVE>
        PFAn
        X

DEFINE  CMD commandname commands
DEF     CMD 0
        PREFIX pppp tt mmmmmmmm
        ENTER commands
        INPUT commands
```

*n* is a function key number (1 to 24). *commands* is a string of one or more editor commands (or macros),

separated by the command delimiter character (normally semicolon ";").

Abbreviations: DEF (for DEFINE), <RETR> (for <RETRIEVE>), PRE or PREF (for PREFIX), ENT (for ENTER), INP (for INPUT).

You can customize the operation of the program function keys by using the DEFINE command, which defines a function key as equivalent to any string of Editor commands. Any function keys defined in this way override the standard default function key definitions.

The DEFINE command can also change the definition of the X command, redefine command and macro names (by the CMD option), define prefix area commands (by the PREFIX option), specify additional commands to be done whenever the Enter key is pressed, and specify a macro to process text entered during Input Mode.

**Defining Function Keys and the X Command:**

*n* in the Fn or PFn parameter is an actual program function key number (1 to 24). Fn and PFn mean the same thing.

*n* in the PFPn (primary PF key) or PFAn (alternate PF key) parameter is a program function key number (1 to 12), which maps to either PF1-12 or PF13-24 on your actual keyboard. (The parameters PFPn and PFAn are rarely used, and should be avoided for most newer types of workstations.)

*commands* is a string of 1 or more Editor commands, separated by the command delimiter character (normally semicolon, (;)).

The DEFINE command must be the only command on the input line. An X command string must not contain an X command. Function keys may not be available on some non-3270-type workstations, but the DEFINE command can make the X command equivalent to a string of Editor commands. Then typing the command X causes the specified sequence of commands to be executed. Also, the command Xn (where n is 1 to 24) can be used to execute the command string defined for PFn, on any type of workstation.

If the command string is omitted, the function key or X command is made undefined.

You can define a function key to retrieve previous command lines with the <RETRIEVE> (abbr. <RETR>) parameter. Example: define pf24 <retrieve>. Up to 10 command lines are saved. It is different from other function keys in the editor, in that the command area is ignored. Press the key repeatedly to retrieve successively older commands.

You can define a function key as a tab operation (moves the cursor to the next tab position, as defined by the TABIN command), by using the <TAB> parameter. Example: define pf4 <tab>.

**Redefining Command Names: DEFINE CMD xxx yyy**

The form DEFINE CMD xxx yyy, where xxx is a command name and yyy is any character string, tells the editor to change the command "xxx tttt" (entered by the user) to "yyy tttt" before processing it. The original command name xxx is replaced by yyy.

**Example:**

```
define cmd list print
```

Then "list abc" (entered by the user) is processed as "print abc".

This replacement is done before searching the editor's command table, and before passing the command to

Rexx. If yyy is omitted, then xxx is considered to be undefined to the editor (and the command will be passed to Rexx). These DEFINE commands are cumulative. A table of keyword replacements is built in a 2K area. DEFINE CMD 0 clears the table.

*xxx* can be specified as fff-ggg, where fffggg is the full command name and fff is the minimum abbreviation.

*Notes on command replacement*:

- If a command name appears twice in the replacement table, the last matching entry (latest definition) is used.
- xxx should not start with % or /, since replacement is not done on commands starting % or /.
- If yyy is longer than xxx, long parameters may be truncated.
- Command replacement is not done on commands from Rexx macros.
- Command CREP OFF turns off replacement.
- Be careful when using macro or program names that contain digits or special characters, since the first non-letter is taken by the editor as the end of the command name. The leading letters may be subject to command name replacement. (See the next note.)
- If the last character of yyy is ! (exclamation point), it is changed to a blank when replacement is done. This can act as a separator between the macro name and the parameters when the original editor command does not require a blank after the command name (e.g. INSERT).

  **Example:**

```
def cmd i-insert Xi!
```

  causes i* to become xi * (where XI is a macro).

**Defining Prefix Area Commands (PREFIX):**

The form DEFINE PREFIX pppp tt mmmmmmmm defines a prefix area command name pppp, of type tt, to be processed by macro name mmmmmmmm. pppp (1 to 4 characters) is the name of the prefix command. tt (1 or 2 characters) defines the type of prefix command (see below). mmmmmmmm (1 to 8 characters) is the name of the macro that performs the prefix operation. The macro name should not be the same as the name of a normal editor command. The keyword PREFIX may be abbreviated PREF or PRE.

The standard prefix commands (I, D, C, MM, etc.) are predefined. You would use the DEFINE PREFIX command only to add your own prefix operations, or to undefine or rename existing ones. When parameters tt and mmmmmmmm are omitted, the prefix command pppp is made undefined. Refer to topic "Prefix Area" earlier in this chapter for more information on the prefix area and prefix commands.

The types tt for DEFINE PREFIX are:

1 Non-paired command, with no target required (e.g. In, Dn)
1T Non-paired, requiring a target (e.g. Cn, Mn)
1L Same as 1, except the operation is done last, after other prefix operations (e.g. /).
2 Paired, no target (e.g. DD, >>)
2T Paired, requiring a target (e.g. CC, MM)
2L Same as 2, except the operation is done last, after other prefix operations.
F Target, of type "following" (e.g. F)
P Target, of type "preceding" (e.g. P)

For example, suppose you wish to use prefix command XX to print a group of lines. XX is entered in the prefix area of the first and last lines of the group. You would define it by:

```
define prefix xx 2 myxxmac
```

"myxxmac" is the name of the macro to be invoked and "2" is the type of prefix operation (in this case, a "paired" command). You would have to write the editor macro in Rexx, and store it as file MYXXMAC.MAC. The macro would do whatever is needed to print the lines. It would be called by the editor as "MYXXMAC n1 n2 0 0 XX op", where n1 and n2 are the line numbers of the first and last lines of the group; XX is the prefix command name; op is the parameter (if any).

**Defining Commands for the ENTER Key**

The command DEFINE ENTER xxx, causes the command string xxx to be done whenever the Enter key is pressed in full-screen (FS) mode. This is in addition to the normal effects of the Enter key. The commands xxx are done after any prefix operations or command-area commands. The command DEFINE ENTER (with xxx omitted) cancels this.

**Defining a Macro to Process Input Mode Text**

The command DEFINE INPUT xxx, where xxx is normally the name of an editor macro, causes xxx to be invoked to process lines added during full-screen (FS) Input Mode. The command DEFINE INPUT (with xxx omitted) cancels this. For example, Power Input Mode (the POWERINP macro) is supported by using DEFINE INPUT $FINP. See the comments in the $FINP macro (file $FINP.MAC).

The macro xxx is invoked with the following parameters:

| | |
|---|---|
| xxx NOTINP | Just before screen display for Command Mode. |
| xxx INPSTART | Just before screen display for Input Mode. |
| xxx n1 n2 | After lines n1 thru n2 have been added to the file by Input Mode. This is done when Enter or a function key is pressed in Input Mode, if 1 or more lines were entered. The macro is invoked before any prefix operations are done. Input Mode is turned off before the macro is called; the macro should turn Input Mode back on if it wants Input Mode to continue. |

For more information see the topic "Defining Prefix Macros" later in this chapter.

**Examples:**

```
DEFINE F13 TOP
```
Define the F13 key as equivalent to a TOP command.

```
DEFINE X LOCATE ABC;CHANGE/ABC/1234/
```
Define the X command as equivalent to 2 successive commands: LOCATE ABC and CHANGE/ABC/1234/.

```
DEF CMD AT-TRIB /ATTRIB
```
Any ATTRIB command (minimum abbreviation AT) entered by the user is passed to Rexx as MUSIC command /ATTRIB. It is not processed as the normal editor ATTRIB command.

```
DEF CMD LIST
```
Makes the LIST command name undefined to the editor. It will be passed to Rexx. (Note: abbreviation LI is still defined as the editor LIST command.)

```
DEFINE CMD TR-OUVER LOC
```
Defines TROUVER (minimum abbreviation TR) as an alias for the LOCATE command.

("Trouver" is French for "find").

Refer to the topic "Advanced Features" for examples of the DEFINE command and how to make definitions automatic whenever you use the Editor.

```
DELCHAR [x  ]
DELC    [OFF]
```

This command defines the delete character for use with the MINSERT and MDELETE commands. *x* is the delete character to be used. It must be a special character, not a letter or a digit. If *OFF* is specified, the delete character is made undefined. If the command is used without a parameter, the current delete character is displayed. The default delete character is ":" (the colon).

When a file is saved by an Editor SAVE, FILE or EXEC command, the Editor looks for lines with the delete character in column 1 (followed by a blank or another delete character). If such a line is found, this probably means that you forgot to use MDELETE. The Editor issues a warning message: `LINE WITH DELETE CHARACTER FOUND. CONTINUE SAVE?` Answer Y or YES if you wish the file to be saved as is. Answer N or NO if you wish the save not to be done. You can then use MDELETE and redo the save operation. (If editor messages are currently suppressed by the MSGS OFF command, this verification prompt is not done.)

```
Form 1:  DELETE [n][,(logical expression)]
         DEL
Form 2:  DELETE /string/[,(logical expression)]
         DEL
Form 3:  DELETE.
         DEL.
```

The DELETE command removes lines from the file, beginning with the current line. (If you want to delete an entire file see the PURGE command.)

With form 1, it deletes *n* lines from the file, beginning with the current line. With form 2, it deletes lines down to, but not including, the line containing *string*, starting with the current line. Logical expressions are optional. With form 3 (DELETE.) the dot at the end of the command signifies that a group of lines previously marked with the MARK command are to be deleted.

**Parameters:**

n — It specifies the number of lines that are to be deleted starting with the current line. If * is specified, the deletion will apply to the remainder of the file. The default value of *n* is 1.

(logical expression) — If it is specified, only the lines for which the logical expression is true will be deleted.

string — Character string which signifies the end of deletion. The current line and all lines after the current line down to, but not including, the line containing *string* are deleted.

**Examples:**

```
DEL                 Removes the current line from the file.
DEL10               Deletes 10 lines from the file starting with the current line.
DELETE /STOP/       This deletes the current line and all lines after it down to, but not
                    including, the line containing the string STOP.
DELETE *,(/TEST/(F,C73))

                    This deletes all lines in the file with the character string TEST in
                    columns 73 to 76, from the current line to the end of the file.
```

See also DELETEL, MDELETE, and UNDELETE.

**DELETEL (logical expression1)[,(logical expression2)]**
**DELL**

The current line and all following lines are deleted, down to but not including the next line which tests true against the *(logical expression1).* If no such line exists, an error message is displayed and the file is not changed.

If the *(logical expression2)* is specified, only those lines for which the *(logical expression2)* is true will be deleted. Parentheses are required around both logical expressions.

**DELIM [x]**
**OFF**
**ON**

The DELIM command changes the command delimiter character to the character specified by *x.* The delimiter character *x* may be any non-alphanumeric character. It separates editor commands entered on the same line.

If the DELIM command is not used, the command delimiter character is assumed to be a semicolon (;) by default. If the command DELIM is entered with no parameter, then no character is the delimiter character, and only one command may be entered on a line.

The command DELIM OFF disables the delimiter character, and only one command can subsequently be entered on a line. However, the original delimiter character (if any) is remembered, and can be enabled later by the command DELIM ON. This makes it possible to temporarily undefine the command delimiter (for example, in order to use the INSERT command to insert text that may contain the delimiter), without knowing what the original delimiter character is. If DELIM ON is used after a DELIM command with no parameter, nothing is changed.

**Example:**

```
delim $
l xaz$c/a/y/$l abc
```

```
delim off
insert The $ in this line is not a delimiter
delim on
last$insert xyz
```

**DOWN [n]**
**DN**

The DOWN command is the same as the NEXT command. Refer to the description of the NEXT command.

**DOWNPAGE**
**DOWNP**
**DNP**

This command is used only in 3270 full-screen mode. It shifts the screen display to the next screen (page) in the file (towards the end of the file). The first line displayed on the new screen will be the line after the last line on the current screen. This command corresponds to F8 by default.

**DOWNWINDOW [n]**
**DOWNW**
**DNW**

The DOWNWINDOW command is used only in 3270 full-screen mode. It shifts the screen display (window) towards the end of the file. The parameter *n* is the number of lines by which the screen is to be shifted. If *n* is omitted, 6 lines is assumed.

**DUP [n]**

This command causes the current line to be duplicated *n* times. If *n* is not specified, it is assumed to be 1.

```
ECHO  [string]
      [NAME  ]
      [MSG   ]
```

This command causes the current line or a specified text string to be displayed in the command area of the screen if the user is in the full-screen mode. The file name or the screen message area can also be displayed. This command is mainly used in conjunction with function keys. The Editor will ignore this command if the user is not in the full-screen mode.

If no parameter is specified, the current line will be displayed from column 1 to the rightmost end of the window setting in the command area. If *string* is specified, *string* will be displayed in the command area. If *NAME* is specified, the file name from the title line of the last full-screen display will be displayed in the command area. *(NAME* can be abbreviated to *NAM* or *NA.)* If *MSG* is specified, the contents of the message area at the bottom of the last full-screen display will be displayed in the command area. In any case, if the length of the line to be displayed is longer than the command area, the line will be truncated to match the length of the command area. The cursor is also placed in the command area. The position of the line pointer is not changed by this command.

Command delimiter characters which appear after the ECHO command are not honoured, provided the ECHO command is not preceded by other commands on the same line. This allows the parameter to be a series of Editor commands.

**Spacing:** One blank should be present between the command name and *string.* All other blanks are part of *string.*

**Example:**

Enter the following in the command area:

```
DEF F8 ECHO MERGE F1 001 002;LOC ABC
```

Press the F8 key and the following is displayed in the command area:

```
MERGE F1 001 002;LOC ABC
```

Change the line numbers (001, 002) accordingly and press ENTER key to execute the MERGE and LOC commands.

```
END <n>
```

Same as QUIT command. The END command terminates the Editor session without saving a copy of the changed file.

If you have made changes to the file but have not issued a FILE command to make the changes permanent, you will be prompted for permission to end the edit session. Enter YES or Y to end the session, or NO or N to cancel the END operation and continue editing.

See QUIT for description of *n* parameter.

```
ENQ
```

This macro is used to prevent more than 1 simultaneous edit of a file. Invoke once only at the start of an edit session. Subsequent invocations will be ignored.

*Note:* Warning: This macro enqueues on only the first 22 chars of the file name (after "userid:" part has been added if not already there). So for long names, it could erroneously report that the file is already being edited.

```
EXECUTE  [name]
EXEC
EX
```

This command saves the changed file in your library under the name *name*, and then automatically requests MUSIC to execute the program contained in that file in a way equivalent to the user typing the MUSIC command "EXEC name". If *name* already exists in the user's library, the Editor will prompt the user whether to replace the existing file or not. If YES (or Y) is replied, the edited file will replace the previously existing file, and the EXEC will be done. Otherwise, no operation will be done. The prompt is not done if replacing an existing file which was read by EDIT at the start of the edit. File attributes can also be specified after the file name, as in the FILE command. If no name is specified on the Editor EXEC command, the name on the original EDIT command or on the last NAME command will be used. If desired, column number limits (as on the FILE command) may be specified before *name*. The parameter *name* may be /INPUT.

If this command is used in the User Data Set version of the Editor, a file name must be specified with the command. The Editor will try to replace the named file by the temporary updated version of the UDS file being edited. If the replacement is successful, a request to execute the program in the named file will be made to MUSIC. The original UDS file is not updated, and the edit is terminated.

```
FF *
FF .
```

The FF macro formats part or all of your file using MUSIC SCRIPT. The lines to be formatted can contain SCRIPT control words such as .SP and .PA, which will be honoured. The margins are from columns 1 to 65 by default, but can be changed by the MARGINS command. There is also a paired prefix area operation called FF. See the topic "Prefix Area" earlier in this chapter. One of the following parameters must be included with the FF command:

* - formats the entire file
. - formats only the marked lines (MARK command)

UNFF undoes the immediately preceding FF command or FF prefix operation.

*Note:* Another way of formatting text is to use the WW prefix area command. See the topic "Prefix Area" earlier in this chapter.

See also UNFF and MARGINS commands.

```
FILE  [m] [n] [name] [PRIV] [XO] [CNT]
              [*   ] [PUBL]
                    [SHR ]
```

The FILE command is the usual way of terminating an Editor session and saving the updated file. This command saves or replaces the updated file directly into your library.

**Parameters:**

| | |
|---|---|
| name | Specifies the name for saving (or replacing) the updated file in your library. If *name* already exists in your library, the Editor will ask whether to replace the existing file or not. If the user replies with YES (or Y), the edited file will replace the previously existing file. No save operation will be done if the user's reply is not YES. Note that if *name* is not specified, the Editor will use the name which was specified on the EDIT command or on the last NAME command, or prompt you to enter a file name if no name is known to the Editor. |
| m n | These parameters specify the first and last column numbers, respectively, that will be saved for each line. Remaining columns are filled with blanks. If only one number is supplied it is assumed to be n and m will be assumed to be 1. The column numbers must appear before the "name" parameter. If omitted, each line will be saved in full. When you specify column numbers, the editor asks for verification before doing the command. Enter yes (or y) to allow the command to continue. Enter anything else to cancel the command. (If editor messages are currently suppressed by the MSGS OFF command, the verification is not done.) |
| PRIV | The file will be made private. |
| PUBL | The file will be publicly readable and in the common index. |
| SHR | The file will be publicly readable but not in the common index. Other users must specify the owner's userid in order to access the file, as in userid:filename. |
| XO | The file will be execute-only. |
| CNT | The system will maintain a usage count for the file (not for UDS files). The count is increased by 1 each time the file is opened. It is displayed by the "ATTRIB filename" command in *GO mode. Note that using the editor to modify the file resets the count to zero, since the editor always recreates the file. |

## Notes on File Attributes

If an existing file is edited and then saved back by using the SAVE, FILE or EXEC command without specifying a file name or attributes, all the original attributes of the file are preserved. This technique should be used when it is desired to change a file without changing its attributes. In other cases, PRIV is used as the default attribute.

**Examples:**

`FILE 1,72,FILEAB`
saves columns 1 to 72 inclusive for each line in the updated file, under the name "FILEAB".

`FILE`
replaces the original file with the updated version of the file.

`FILE FILE1,PUBL,XO`
saves the updated file as a public execute-only file in your library under the name "FILE1".

**Messages:**

`INVALID COMMAND, or INVALID OPERAND`
Something is wrong with the command name or the parameters.

`FILE IS EMPTY`
This is a warning message telling you that the file being edited is empty (has no lines). The FILE command is allowed (not for UDS files).

`INVALID FILE NAME, PLEASE TYPE A NEW NAME`
The file name specified in the command is incorrect. Check that the file name is of the form NAME or userid:NAME, where NAME is 1 to 17 characters long and userid is the owner's userid. Valid characters are letters (A to Z), digits (0 to 9), and the special characters $ # @ _ and period (.). The first character of NAME must not be a digit or period. Case is not significant in file names, since lower case letters are automatically converted to upper case. The NAME part can be preceded by directory names, as in ABCD:TEST\NAME or ABCD:PROGS\TEST\NAME. The special names /INPUT, /HOLD and /REC are also allowed in most cases.

`TYPE NAME OF FILE TO BE REPLACED OR CREATED`
No name is currently associated with the edit session, and you did not specify a file name on the command. Enter the name of the file you wish to replace (if the file already exists) or create (if this is a new file).

`ONLY COLUMNS n TO m WILL BE SAVED. TYPE YES OR NO.`
You have used the column number option on the command (1 or 2 column numbers were specified as the very first parameters). Type yes (or y) if you wish the command to proceed. Otherwise type no (or n). (If editor messages are currently suppressed by the MSGS OFF command, this verification is not done.)

`*EXCESSIVE OUTPUT, JOB TERMINATED`
The editor has attempted to write too much data to the holding file (unit 10). This is usually caused by a command such as FILE /INPUT, RUN or SAVE (10) when the file being edited is too big for unit 10 or the /INPUT file. Specify a file name instead.

`NOT YOUR LIBRARY ...OPERATION NOT DONE`
The file name you have used is the name of a file belonging to another user (a userid different from your userid). You cannot replace that file. Choose a different name.

`NAME ALREADY USED BY SOMEONE ELSE`
The (public) file name you have used is already used by someone else, or is a name which the system will not let you use. Choose some other name.

`UNABLE TO OPEN TEMPORARY FILE`
The most likely cause of this error is that the MUSIC Save Library is full, and there is no room to save your file. Wait a few moments, then retry the save. While still in the editor, you can use the PURGE command to delete any of your files which you no longer need. If the problem persists, contact the User Consultant or the MUSIC Systems Administrator, who will try to make space available. Keep trying the save.

`ERROR WRITING TO TEMPORARY FILE`
An error occurred while writing the file to disk. Try the save again. If the error persists, notify the User Consultant.

`UNABLE TO CLOSE -- FILE NOT SAVED`
The editor was unable to complete the save operation to the specified file. Try the save again. If the

error persists, notify the User Consultant.

`FILE IS TOO BIG - SIZE LIMIT EXCEEDED, or:`
`FILE WOULD CAUSE LIBRARY SPACE LIMIT TO BE EXCEEDED`
The file you are trying to save exceeds one of the space limits associated with your userid, or would cause your total space limit to be exceeded.

`FILE IS TOO BIG - CANNOT GET MORE SPACE`
The file could not be saved because too many extents were required, or there was not enough free space in the Save Library. Try the save again. If the error persists, notify the User Consultant.

`YOUR USERID CANNOT CREATE PUBLIC FILES - FILE NOT SAVED`
Your userid has the "private-only" restriction, meaning that you cannot save a public file (i.e. one that other users can access). Redo the save, without specifying the PUBL or COM option.

`FILE EXCEEDS SIZE OF DATA SET BY n RECORDS`
The user data set is too small to hold all the records you have attempted to SAVE or FILE. Either delete some lines from the file being edited, or increase the size of the data set and redo the edit.

`DATA SET IS READ-ONLY`
You attempted a SAVE or FILE operation to a user data set which was defined as read-only (no writes are allowed to the data set). Redo the edit, specifying OLD on the /FILE statement for the data set.

`I/O ERROR WHILE WRITING TO THE DATA SET`
A disk input/output error occurred during a SAVE or FILE operation to a user data set. Try the command again. If the problem persists, notify the User Consultant or the MUSIC Systems Administrator.

`NO BUFFERS AVAILABLE`
This message indicates an internal problem with the editor. If the problem persists, notify the User Consultant or the MUSIC Systems Administrator.

`LINE WITH DELETE CHARACTER FOUND. CONTINUE SAVE?`
This message indicates that there are lines which start with the delete character followed by blanks, or start with 2 delete characters. It gives you a chance to delete those lines, if you want, before the file is saved. The delete character (normally ":") is defined by the DELCHAR command. Type yes (or y) if you want the file saved as is. Type no (or n) if you want to stop the save operation (this gives you a chance to fix the file and then redo the save). If editor messages are currently suppressed by the MSGS OFF command, this verification prompt is not done, and the file is saved as is.

See also QUIT, SAVE, EXEC, RUN, NAME, STORE, OKREPL.

**FILL [FLIP]**
**[F ]**

The FILL command applies only to 3270 full-screen mode. It causes blanks to be displayed at the end of each field of the screen, rather than null characters. An alternate name for the FILL command is NONULLS. The opposite of this command is NOFILL, which is the default method of display. Refer to the section on full-screen mode for more information.

The FLIP parameter reverses the current setting for the command; i.e., FILL FLIP causes blanks to be displayed at the end of each field if null characters were previously displayed, or causes null characters to be displayed if blanks were previously displayed.

```
FIND  [string]
F
```

The FIND command searches the file, starting with the line after the current line, for a line beginning with *string.* The search continues down the file until the first match, or until the pointer has been moved past the last line of the file (the message *EOF is displayed). If issued after the pointer is past the end of the file, an automatic TOP is performed before the search begins. If VERIFY status is in effect, the line found is displayed at the workstation.

If *string* is not specified, the same string as specified in the last used FIND, LOCATE, UFIND, ULOCATE, HUNT, or SEARCH command is used. (The UFIND command is similar to the FIND command but searches upwards through the file.)

**Spacing:** One blank should be present between FIND and *string.* All other blanks are considered to be part of *string*, which extends to and includes the rightmost non-blank in the command line.

**Pointer:** Set to the found line, or beyond the end of the file if no line is found which begins with *string.* (Message *EOF is displayed in this case).

**Example:**

Before:
```
SAMPLE LINE          <--Pointer
ANOTHER LINE
LINE THREE
```
Command:
```
F LINE
```
After:
```
SAMPLE LINE
ANOTHER LINE
LINE THREE           <--Pointer
```

```
MUSIC/SCRIPT Form:

FLAG  SCRIPT [COL=n]
FLA
```

This form of the FLAG command is particularly useful when using the Editor to make modifications to a file that is to be used by the MUSIC/SCRIPT program. (The MUSIC/SCRIPT program is described briefly in *Chapter 1. Introduction* of this manual and in more detail in a separate *MUSIC/SP Mail and Office Applications Guide*.) This command will cause all modified lines to be date stamped in columns 73 through 77, with the current date in the form YYDDD. Deletion text to be added to the front of all deleted lines when flag mode is in effect (see the general form of the FLAG command).

This command automatically issues the following Editor commands:

```
TOP
TEXT SCRIPT
ZONE 72
WINDOW 1,72
```

The specification of the COL=n parameter can be used to put the date stamp starting in column number *n*

rather than 73. The generated ZONE and VERIFY commands would then be changed accordingly.

**Pointer:** The pointer is moved to the top of the file.

**General Form:**

```
FLAG  [flgtxt] [DEL=deltxt] [COL=n]
FLA
```

This command is the general form of the FLAG command. It puts the Editor into *flag mode*, which causes lines which are subsequently changed, added or deleted, to be identified with a special character string called a *flag*. The flag is normally placed towards the end of the line. This makes it possible to maintain a record of all modifications to a file.

The parameter *flgtxt* is a 1 to 10-character string, optionally enclosed in single quotes, to be used to identify all new, changed or *logically* deleted lines when flag mode is in effect. The default for this parameter is the current date in the format YYDDD (5 characters long) if no previous FLAG command was used, or else whatever was specified on the previous FLAG command.

The parameter *deltxt* is a 1 to 8-character string, optionally enclosed in single quotes, to be added to the front of all "logically" deleted lines when flag mode is in effect.

The parameter *n* is the starting column number where *flgtxt* is to be placed in each line. The default for the first FLAG command is COL=73.

When the flag option is turned on, the Editor automatically issues the following commands:

```
TOP
ZONE n-1
WINDOW 1,n-1
```

The *n* in the above commands is the number given in the COL=n parameter, or 73 if not previously defined.

When flag mode is in effect, any new lines added by INPUT, INSERT, etc., will be flagged. The flagging operation is performed by placing *flgtxt* at the specified column. The CHANGE command only flags a line if a change was actually made.

The DELETE command only *logically* deletes the line, by inserting *deltxt* at the beginning of the line and *flgtxt* at the appropriate column. A line will not be logically deleted if (1) it already begins with the deletion text, in which case it is left as is, or (2) the line has the current flag, in which case the delete is really done. Case (1) takes precedence over (2).

A line is considered to be logically deleted if it has *deltxt* starting in column 1. In flag mode, the logically deleted lines are generally transparent to the Editor. They are not displayed by PRINT n, are skipped by UP and NEXT, and are not seen by commands which search for character strings. However, the commands TOP, LAST, = and SIZE have the same effect with or without flag mode.

**Pointer:** The pointer is moved to the top of the file.

**FLIP [x]**
**FL**

This command defines the first non-blank character following the command as the *flip* character. It must not be a letter or digit. This flip character may then be entered after a command name or abbreviation, and has the effect of reversing the last preceding BRIEF or VERIFY command, for the current command only.

A flip character may be used with the following commands: ADD, BLANK, CHANGE, CHANGEL, DELETE, DELETEL, FIND, HUNT, LAST, LOCATE, UFIND, ULOCATE, LOCATEL, NEXT, OVERLAY, SEARCH, SEARCHL, SPLIT, UP. If a flip character is used in combination with a column number specification Cn, the flip character must follow the last digit of the column number.

If a flip character is used while the Editor is in BRIEF status, the number of characters displayed is as defined by the last WINDOW command issued.

If no non-blank character is specified, the FLIP option is turned off. At the start of the Editor session, the option is off (i.e. there is no flip character defined).

**Spacing:** Blanks between the command and the first non-blank character are ignored.

**Example:**

In this example, the pointer would be moved three lines down, as specified by the NEXT command, but typing of the line reached is suppressed.

```
VERIFY
FLIP *
N* 3
```

**FORMAT**

This macro is used to format all the text in a file while remaining in the edit session. The SCRIPT program is used to perform the formatting with default SCRIPT control words incorporated to fit the text in 72 columns. If you wish, you can include your own control words to override the default settings.

The original text (unformatted) is stored temporarily in a holding file at the time you issue the FORMAT macro. You can restore the file to its original form by using the UNFORMAT macro.

This macro is not recommended for large files as the process is time consuming. It is ideal for formatting mail text when you are using the editor in the MAIL program.

See also FF, MARGINS, and PREFIX.

```
FS  [NOPFK]
    [NOPF ]
    [PF12 ]
    [PF24 ]
    [n    ]
```

The FS command begins full-screen mode, which applies only to 3270-type workstations or ASCII terminals that accept 3270 data streams. Refer to the section on full-screen mode for more details. The opposite of this command is NOFS.

Full-screen mode is automatically assumed at the beginning of the edit, whenever the workstation can support it.

The optional parameter *NOPFK* should be used only if your 3270-type workstation does not have any program function (PF) keys. It tells the Editor to put the screen cursor into the command area whenever the screen is displayed, thus facilitating the entry of commands.

The optional parameter PF12 or PF24 is used to inform the Editor about the actual number of function keys on the workstation being used if the number was not correctly set at sign-on time. This is important for the Editor in order to set the default function key definitions correctly on the workstation being used. PF12 means that the workstation being used has 12 function keys, while PF24 means that there are 24 function keys.

The command FSn begins full-screen mode, and defines the total number of lines on the screen. n can be a number from 7 to the actual screen size (usually 24, 32, or 43). When 3270 full-screen mode is being used from a remote 3270, or from an ASCII terminal simulating a 3270, it can take an appreciable amount of time for the screen to be displayed, especially when connected at 300 or 1200 baud. In these cases it is sometimes useful to define the screen as containing fewer lines than it actually does.

```
GETV name
```

The macro GETV displays the value of a SETV name. Refer to the SETV command.

```
HELP [command-name] [list-of-topics]
HE
```

This command is used to obtain information about a particular Editor command, or about the Editor in general.

If no parameter is specified, the command gives general information about the Editor, such as a one-line description of common commands, Editor concepts, etc.

If *name* is specified, where *name* is the name or abbreviation of an Editor command, information about the command is displayed.

To access MUSIC's general help facility from the Editor, use a slash (/) preceding the HELP command. Enter "/HELP" or "/HELP topicname" from the command area of the Editor. (Without the "/" you will receive help on the Editor and not MUSIC's general help facility.)

**Examples:**

```
HELP          obtains general information about the editor.
HELP MOVE     explains the use of the MOVE command.
HELP TOPICS   gives a list of all available topic names
/HELP COPY    places you in MUSIC's general help facility and gives information about
              MUSIC's COPY command (not the Editor's COPY command).
```

**HEX**

This command causes the lines to be displayed in *hexadecimal* format. Hexadecimal format causes each character to be displayed as two hex *digits* 0 to F. For example, a blank is hexadecimal 40 and the number 9 is F9.

The Editor will display 32 characters per line in this mode. This command also temporarily turns off the screen mode if it is on at the time the command is issued; screen mode is resumed when the ALPHA command is used.

The commands BOTH or ALPHA can be used to tell the Editor to use other output representations.

**HUNT [string]**
**H**

The HUNT command combines the effect of a TOP command followed by a FIND. It searches the entire file for the first line beginning with *string.* If in VERIFY status, the found line is displayed at the work-station. If there is no line beginning with *string*, the message *EOF is displayed.

If *string* is not specified, the same string as specified in the last used FIND, LOCATE, UFIND, ULOCATE, SEARCH, or HUNT command is used.

**Pointer:** Set to the found line, or beyond the end of the file if no line is found which begins with *string.* (Message *EOF is displayed in this case.)

**Spacing:** One blank should be present between the keyword or abbreviation and the first character of *string.* All other blanks are part of *string*, which extends to the last non-blank in the command line.

```
INPUT [END=xx]
INP
```

The INPUT command changes the mode mode of operation from edit to input. All lines typed after the INPUT command are placed, sequentially, after the current line. Input mode can also be started by the INP prefix command.

In 3270 Full-Screen mode, input mode provides a large area of the screen for you to type lines. If you fill up the area and wish to continue typing more lines, press the Enter key. When you are finished typing lines, press the Enter key twice, or press a program function key such as F12, to end input mode.

Outside of 3270 Full-Screen mode, you enter a blank or null line during input mode, to end input mode and return to edit (command) mode. The Editor switches to edit mode with the pointer at the last line entered (not counting the blank line which is not saved).

If the command is given at the very beginning of the Editor session, or immediately after a TOP command, the new lines are placed before the first line of the file.

The parameter END=xx, if used, causes the Editor to recognize *xx* as the input mode terminator, rather than a blank line. This option cannot be used in 3270 full-screen mode. *xx* may be one or two characters long, and any nonblank characters may be used. Input mode is terminated by entering a line with *xx* starting in column 1, followed by blanks. This *xx* line is not saved. The *END=* parameter is useful when it is required to enter blank lines while in input mode.

See also MINSERT and POWERINP.

```
INSERT  [string]
I
```

INSERT places a new line, containing *string*, after the current line. Note: If the command is given immediately after TOP, the new line is inserted before the first line of the file. A blank line can be inserted in the file just by typing INSERT.

CAUTION: The use of tab characters with this command may not have the desired effect. For example, the sequence "I tx", where "t" is the tab character, will put "x" in column 8 if the first tab location was defined to be column 10.

**Spacing:** One blank must be present between the command name or abbreviation and *string.* Any other blanks are part of *string.*

**Pointer:** Set to the inserted line.

**Example:**

```
Before:   SAMPLE LINE          <--Pointer
          ANOTHER LINE
Command:  I YET ANOTHER
After:    SAMPLE LINE
          YET ANOTHER          <--Pointer
```

**ANOTHER LINE**

**JOIN [/string/]**
**JO**

The JOIN command may be thought of as the opposite of the SPLIT command. It joins the next line to the end of the current line to form one line.

The second line is added after the last nonblank character in the "ZONE" part of the first line. A specified character string (or a single blank if a string is not specified) is placed between them. Only the "ZONE" parts of the lines participate (refer to the ZONE command). If the joined text is too long for one line, a warning message is displayed and the excess is left on the second line.

A single blank is used if */string/* is not specified.

**Example:**

```
Before:   line1
          line2          <--- current line
          line3
Command:  join /+++/
After:    line1
          line2+++line3  <--- new current line
```

**KEYS**

The KEYS Editor macro allows you to change your function key definitions.

**LANGUAGE [language]**
**LANG**

The LANGUAGE command specifies the national language to be used for the remainder of this workstation session. It is similar to the MUSIC LANGUAGE command. The national language setting affects the language used in messages from some applications. It also turns Double-Byte Character Set (DBCS) mode on for the current Editor session, if the requested language is a DBCS language, such as KANJI.

*language* is the language name, or is DEFAULT to request the system default language. In most cases, the first 3 characters of the name can be used as an abbreviation. Not all languages are supported or installed at all sites. Some language names are:

| | |
|---|---|
| DEFAULT | (the system default language) |
| ENGLISH | |
| FRENCH | |

KANJI (JAPANESE)
PORTUGUESE
SPANISH

See also: DBCS.

**LAST**
**LA**

This command positions the line pointer to the last line of the file.

**LEFT [n]**
**LE**

The command LEFT shift the window display left a specified number of columns. This is equivalent to the command WINDOW LEFT n. If *n* is omitted, the window is shifted the maximum number of columns possible.

See also RIGHT, WINDOW and ZONE.

**LIST [filename[,m][,n]]**
**LI**

This command is used to display the contents of a file, or part of a file, or the entire file being edited. LIST is often used in conjunction with the MERGE command.

Line numbers *m* through *n* of the file *filename* are listed. If the *m* or *n* parameter are omitted the entire file is listed. If the *n* parameter is omitted or is too big, then the listing stops at the end of the file. The LIST command without any parameters displays the entire file being edited. Line numbers will also be displayed if line numbering is in effect. Line numbering is requested by using the NUMBER command.

The file name may be /INPUT (the Input File) or /HOLD (the Holding File).

"(3)" may be specified instead of *filename.* This causes the UDS file on MUSIC unit number 3 to be listed. The file is rewound both before and after the listing operation.

```
LOCATE  [string]
L
```

LOCATE is used to find the first line after the current line which contains *string* anywhere in the line and, if in VERIFY mode, to display the line on the workstation. The search begins with the line following the current line and continues down the file. If *string* does not exist in the file between the line following the current line and the end of the file, the message *EOF is displayed. If the command is given after an end of file an automatic TOP is performed before the search begins. If the command is given immediately after *EOF (bottom of file) or a TOP command, the first line of the file will also be searched for *string*. The operation of LOCATE may be affected by the ZONE setting (refer to the description of the ZONE command).

If *string* is not specified, the same string as specified in the last used FIND, LOCATE, UFIND, ULOCATE, HUNT, or SEARCH command is used.

(The ULOCATE command is similar to the LOCATE command but searches upwards through the file.)

**Pointer:** Set to the line in which the *string* is found, or beyond the end of the file, if it is not found.

**Spacing:** One blank should be present between the keyword or abbreviation and *string*. All other blanks are part of *string*, which extends as far as the rightmost non-blank in the command line.

**Example:**

```
Before:  SAMPLE LINE          <--Pointer
         ANOTHER LINE
         YET MORE
Command: L ER
After:   SAMPLE LINE
         ANOTHER LINE         <--Pointer
         YET MORE
```

```
LOCATEL  [n,](logical expression)
LL
```

The file is searched for the next line which tests true against the *logical expression*. The first parameter is optional and specifies the number of lines to be searched. If it is omitted, the search terminates at the first line which tests true.

If *n* is specified, *n* lines are searched (starting with the one following the current line), each line satisfying the logical expression is displayed, and the pointer is set to the line following the last one searched. If * is used instead of *n*, the search continues to the end of the file and each line satisfying the logical expression is displayed. For example, TOP followed by LL*,(/ABC/) displays all lines containing ABC. In this way the LOCATEL command may be used as a generalized form of the SCAN command.

If the command immediately follows *EOF or a TOP command, the first line of the file is included in the search.

**Example:**

If it desired to search for the next line which contains the strings HOW and DO, but does not at the same time contain both THUD and THUNDER.

```
LL (/HOW/AND/DO/AND.NOT(/THUD/AND/THUNDER/))
```

```
LOG  [n    ]
     [END  ]
     [PURGE]
```

This command is used in conjunction with the Editor restart facility. When used without an parameter, the LOG command forces any log lines being held in main storage buffers to be written to the log file. This ensures that all edit activity prior to the LOG command will be available for restart if necessary.

When used with a number *n* as an parameter, the command defines the maximum number of log lines which will be held in main storage. Initially, the Editor assumes a value of 25 lines, i.e. LOG 25, for 3270-type workstations and 20 lines for other workstations.

The number *n* specified on the LOG command is also the maximum number of lines which may be entered in input mode (in non-full-screen mode) before the Editor gets control. This may be noticeable as a slight pause after each group of n lines entered in input mode.

Using LOG without an parameter does not change the value n.

As explained in the section on the restart facility, additional lines may be lost if the Editor is in input mode or full-screen mode.

If the parameter END is specified, the Editor will clear and close the log file. The log file can then be purged by issuing the command "PURGE @ELOG" to save file space. Use "PURGE @ELOG.x" if your userid has a subcode *(x* being your 1-8 character subcode).

If the parameter PURGE is specified, it is equivalent to specifying the parameter END and the PURGE command as mentioned above. That is, the log file will be cleared, closed, and purged all at the same time.

```
MARGINS n1 n2 [n3]
```

Before using FF or WW, you can use the MARGINS macro to redefine the output margins. For example "margins 5 60" will make an indented paragraph.

The first number *n1* is the left margin column. Formatted text will start in this column.

The second number *n2* is the right margin column. Formatted text will not extend beyond this column.

The optional third number *n3* is the paragraph indent, which can be a positive or negative number of columns. This option applies to the WW prefix command, but is ignored by the FF macro. FF always uses

n3=0. It gives the number of characters to indent the start of each paragraph, relative to the left margin of the rest of the paragraph. A negative number gives "hanging" paragraphs. The default for n3 is 0 or the previous value specified.

See also FF, WW, FORMAT, PREFIX, and MARK.

```
MARK  [?]
MA
```

The MARK command is used to identify the first and last lines of a group of lines. The group may then be operated on by the *dot* forms of the command =, MOVE, COPY, DELETE, CHANGE, CHANGEL, FF, REPEAT, TOUC, TOLC, COPYCOL, SEQ, SORT, and STORE.

Refer to the topic "Marking a Group of Lines" earlier in this chapter.

```
MD
```

This command makes a new directory. Same as the MUSIC command MD.

```
MDELETE  [n]
MDEL
```

This multiple delete command has a dual function. It deletes unused lines left over from the MINSERT command (without the *B* parameter), and can also delete a group of lines.

First, lines in the neighbourhood of the current line which have the delete character in column 1, followed by blanks, are deleted. The delete character (":" by default) is as defined by the DELCHAR command described above. Then, if lines were found with the delete character in columns 1 and 2 (i.e., the line begins with "::"), all lines between the first and second such lines, inclusive, are deleted.

Thus, to delete a group of lines, put delete characters into columns 1 and 2 of the first and last lines of the group, then use the MDELETE function key.

The *n* parameter of the command defines the number of lines above and below the current line to be searched for delete characters. A command such as MDEL 99999 will apply the delete to the entire file (the default value for *n* is 40).

The line after the last line deleted will become the current line.

```
MERGE  filename[,n][,m]
ME
```

This command brings a copy of line numbers *n* through *m* of the file *filename* into the file being edited. If the *n* and *m* parameters are omitted the entire file is copied. If the *m* parameter is omitted or is too big, the file is copied until the end of the file.

The new lines are inserted after the current line, and the last line inserted becomes the new current line. If the preceding command was a TOP, or the MERGE command is given at the very beginning of the Editor session, the lines are placed at the beginning of the file.

The records merged from the file are truncated or filled out with trailing blanks, if necessary, to match the record length of the file being edited.

The file name may be /INPUT (the Input File) or /HOLD (the Holding File).

"(3)" may be specified instead of *filnam.* This causes the UDS file on MUSIC unit number 3 to be used as input for the merge. The file is rewound both before and after the merge. Examples: MERGE (3), MERGE (3),15,50.

```
MINSERT  [n] [B]
MI
```

This multiple insert command causes the insertion of a number of lines following the current line. The first line inserted becomes the new current line.

*n* is the number of lines to be inserted. The default is 10 lines.

If the *B* parameter is specified after the number of lines, then blank lines are inserted. Otherwise, lines with the delete character (as specified by the DELCHAR) in column 1 are inserted.

This command can be used as an alternative to the INPUT mode for entering lines into a file.

```
MOVE i,j,k
MO
MOVE.
MO.
```

This form of the MOVE command allows moving lines by reference to the line numbers of the lines. The block of lines from line *j* to line *k* inclusive is moved to after line number *i.* (Line numbers may be determined by using the "=" command.) The original lines *j* to *k* are deleted. If *i* is specified as 0, the lines are moved to the beginning of the file. If *j=k,* only one line is moved.

Line numbers may also be specified in any of the following forms: *n, *+n, *-n, LAST, LAST-n*, where *n* is a number, * means the current line, and LAST means the last line of the file. Example: MOVE LAST,*,*+3

For the command "MOVE.", used to move a marked group of lines, refer to the topic "Marking a Group of Lines" earlier in this chapter.

**Pointer:** Set to the last line of the inserted section.

**Example:**

Before:
```
A1
A2
A3
A4
A5
```
Command:
```
MOVE 4,1,2
```
After:
```
A3
A4
A1
A2
A5
```

```
MSG xmessage
```

The MSG command causes a message line to be displayed on the workstation. The first character of the message is assumed to be a print control character, i.e. blank for single spacing, zero for double spacing, etc.

*x* is the print control character (usually a blank). Exactly one blank must appear between the command name and the control character. In 3270 full-screen mode, the message is displayed in the message area at the bottom of the screen, if there is room.

```
MSGS ON
     OFF
     NOFILMSG
```

The MSGS command is used to suppress or enable all messages from the Editor. When MSGS OFF is in effect, no messages are sent to the workstation. Also, the output of such commands as LIST and SCAN is suppressed.

MSGS OFF is intended mainly for use in function key definitions, and in Rexx procedures invoked from the Editor (Editor macros). The function key operation or Rexx macro would set MSGS OFF to suppress undesired messages, and set MSGS ON before returning.

The command MSGS NOFILMSG suppresses the file name message and the confirmation message issued by a successful FILE command. This may be desirable when the editor is invoked from a full-screen application, to avoid leaving full-screen mode at the end of the edit.

See also: MSG, REXX

**Example:**

The following defines F1 as a store operation to file ABC. If the file already exists, it is deleted before the store. MSGS OFF ensures that the PURGE command does not produce any messages.

```
DEFINE F1 MSGS OFF;PURGE ABC;MSGS ON;STORE ABC
```

**NAME [name]**
**NA [0 ]**

This command is used to display or change the file name associated with the edit session. *name* may be a file name or the name /INPUT for the Input file.

If no parameter is used, the name of the file being edited is displayed. For a UDS file edit, the volume name is also given.

If 0 (zero) is used as the parameter, the file name is made undefined. Then if a SAVE or FILE command is done, the user will be prompted to enter the file name to be used.

For the normal files, the file name associated with the edit session may be defined or changed by using the new name as an parameter on the NAME command. A subsequent SAVE command, for example, will use the new name.

MUSIC also accepts FNAME, with abbreviation FN, to mean the same as the NAME command.

**Spacing:** The file name, if used, must be separated from the command by at least one blank.

**NEXT [n]**
**N**

This command moves the pointer *n* lines down the file from the current line. If *n* is omitted, 1 is assumed. The new current line is displayed, unless a BRIEF command has been issued previously.

For compatibility with other systems, MUSIC also accepts DOWN (or DN) to mean the same as the NEXT command.

**NOARROW**
**NOAR**

This command removes the effect of the ARROW command.

```
NOCHANGE
```

This command informs the Editor that there are no unsaved changes even though there may have been. For example, if a MERGE has been used within a macro the Editor will allow the QUIT command to be used instead of QQUIT.

```
NOFILL  [FLIP]
        [F   ]
```

This command applies only to 3270 full-screen mode. It is the default setting and is the opposite of the FILL command. NOFILL causes null characters (rather than blanks) to be displayed at the end of each screen field. An alternate name for NOFILL is NULLS.

The FLIP parameter reverses the current setting for the command; i.e., NOFILL FLIP causes blanks to be displayed at the end of each field if null characters were previously displayed, or causes null characters to be displayed if blanks were previously displayed.

NOFILL is the default method of display at the start of an Editor session. The opposite of this command is FILL.

```
NOFLAG
NOFL
```

This command terminates the effect of a previous FLAG command, except that the ZONE and WINDOW settings still apply. However, the flag options are remembered by the Editor, and the original flagging can later be continued by issuing a FLAG command without any parameters.

```
NOFS
```

The NOFS command terminates 3270 full-screen mode. It is the opposite of the FS command. NOFS implies the commands WINDOW* and ZONE*. Refer to the section on full-screen mode for more details.

```
NONULLS  [FLIP]
         [F   ]
```

This command applies only to the 3270 full screen mode. It causes blanks to be displayed at the end of each field on the screen instead of null characters. An alternate name for this command is FILL.

The FLIP parameter reverses the current setting for the command; i.e., NONULLS FLIP causes blanks to be displayed at the end of the field if null characters were previously displayed, or causes null characters to be displayed if blanks were previously displayed.

Null characters are displayed by default at the start of an Editor session. The opposite of this command is NULLS.

```
NONUMBER
NONUM
```

This command is used to suppress the line numbering caused by the NUMBER command. At the start of the Editor session, the line numbering is off.

```
NOSCREEN
NOSCR
```

This command is used to turn off *screen* mode.

```
NOSHOW
NOSH
```

The NOSHOW command removes the text displayed at the bottom of the screen by a previous "SHOW file-name" command, and enlarges the FS screen back to its original size.

```
NOTRAN
```

Normally all output from the Editor is translated to ensure that only printable characters are displayed. The NOTRAN command suppresses this translation.

```
NULLS  [FLIP]
       [F   ]
```

This command applies only to the 3270 full screen mode. It causes null characters to be displayed at the end of each field on the screen. An alternate name for NULLS is NOFILL.

The FLIP parameter reverses the current command; i.e., NULLS FLIP causes blanks to be displayed at the end of each field if null characters were previously displayed, or causes null characters to be displayed if blanks were previously displayed.

At the start of the Editor session, null characters are displayed. The opposite of this command is NONULLS.

```
NUMBER [FLIP]
NUM    [F   ]
       [ON|OFF]
```

The NUMBER command causes line numbers to appear whenever lines of the file are displayed. At the start of the editor session, the line numbering is off. The line pointer is not affected. Either NUM FLIP, NUM OFF or NONUM can be used to turn numbering off.

The records of the file are numbered sequentially, starting at 1.

```
OFF
```

The OFF command is similar to the END and QUIT commands, but also terminates the MUSIC session and disconnects the workstation from MUSIC. It is equivalent to an END or QUIT command, followed by the MUSIC command OFF.

**OKREPL [filename]**

The OKREPL command indicates that the specified file can be replaced without the editor prompting for permission. This affects the FILE, SAVE, EXEC, RUN, and STORE commands. The file name must be specified exactly (except for case), as it will be on the save command, or as in the current file name if the save command does not specify a file name.

The editor remembers only one such file name at a time. A successful save operation has the same effect as the command "OKREPL xxx", where xxx is the name of the file saved. This may nullify a previous OKREPL command.

If OKREPL is used without a parameter, any previous OKREPL is nullified, and subsequent save operations will do normal prompting.

**Example:**

```
okrepl myfile
file myfile
```

**OVERLAY string**
**O**

This command replaces specified characters in the current line with the corresponding characters in *string.* The replacement is position-dependent. If the nth character of *string* in nonblank, then the nth character of the current line is replaced with the nth character of *string. String* may contain up to 79 characters, and for each nonblank character, one character of the current line will be replaced.

This command is applied to the current line. If preceded by a REPEAT command, it is applied to as many lines, beginning with the current line, as were specified in the REPEAT (see description of REPEAT). This command is the complement of the BLANK command. If in VERIFY status, the last line changed by OVERLAY is displayed on the workstation.

**Spacing:** One blank should be present between the keyword or abbreviation and *string.* All other blanks are part of *string.*

**Example:**
Before: **SAMPLE LINE**
Command: **O X**
After: **SAXPLE LINE**

```
POINT  xxxxxxxx [n]
POI             [OFF]
```

This command assigns a 1 to 8 character label (xxxxxxxx) to a line of the file. The label must start with a letter or a digit. Case (upper/lower) is not significant in the label. Periods (.) preceding a label are ignored (e.g. POINT .ABCD 10 is the same as POINT ABCD 10). Later during the edit session, you can go to that line by the command =xxxxxxxx. A label stays with the line, even if the line number changes as a result of inserts, deletes, moves, and copies. A label can start with a digit, but in that case the command to go to it must be =.xxxxxxxx.

Add the parameter *n* to specify a line number, otherwise the current line is labeled. *n* can be 0 to the number of lines in the file +1. OFF undefines the label.

The prefix area command .xxx assigns the label xxx (1 to 3 characters) to the line on which it is entered. See PREFIX for more information.

```
POWERINP
POW
```

The POWERINP command (actually a macro) is similar to the INPUT command, except that you can type continuously without having to press the NEWLINE key or TAB key at the end of each line of input. Power Input Mode is similar to Input Mode, except for these differences:

1. The cursor skips automatically to the beginning of the next line when the cursor reaches the end of a line. This allows you to type continuously without having to watch where the cursor is on the screen.

2. When you press ENTER or an F key while in Power Input Mode, the lines you have just typed are reformatted by the editor. This is called "word wrap". Words that were split at the end of a line are put back together, and text is formatted within the current WINDOW columns. A blank line causes a formatting break (paragraph separator). The formatting is similar to what the WW prefix command would do. See the WW command in the topic "Prefix Area" earlier in this chapter.

3. Pressing ENTER after typing one or more lines of input does not end Power Input. The Enter key causes a formatting break, and Power Input continues. This is useful for entering single lines (like Script control words) that should not be formatted with the rest of the text. To end Power Input, press ENTER twice, or F12 once.

Power Input can also be started by the POW prefix area command. Input starts after the line identified by POW.

If POWERINP is used while not in 3270 full-screen (FS) mode, normal Input Mode is used.

*Note:* To use Power Input with NET3270, document mode (the Alt-E toggle) must be OFF. NET3270 has its own word-wrap feature (Alt-W while in document mode), which makes the editor's Power Input unnecessary.

**Internals:**

Power Input is started by the macro POWERINP, which, among other things, issues the commands DEFINE INPUT $FINP and INPUT. The text typed during Power Input is processed by the macro $FINP. Automatic cursor skip is obtained by the command AUTOSKIP INPUT. For more information, see the DEFINE command and the comments in file $FINP.MAC.

```
PREFIX ON
PRE    OFF
       FLip
       CLear
```

The prefix area, turned on by the command PREFIX ON, is a 4-character modifiable field at the left of each displayed line of the file. It contains ====, or the last 4 digits of the line number if NUM ON is in effect. You can enter various special commands in the prefix area, to do editing operations such as inserting or deleting lines, moving and copying lines, etc.

**Parameters:**

ON is assumed if there is no parameter

OFF reverses the previous setting

CLEAR clears any pending prefix operations

See the topic "Prefix Area" earlier in this chapter for more information.

```
Form 1:  PRINT filename [ROUTE(loc)] [FORMS(x)] [COPIES(n)] [CC  ]
         PRI   *CUR      R            F          C          [NOCC]
               .

                   [PAGELEN(m)]
                    P

Form 2:  PRINT [m] [n]
         P
```

Form 1 of the Editor PRINT command is similar to the MUSIC PRINT command. It schedules the printing of a file to a specified printer. The file name must be the first parameter. Special names *CUR (the data being edited) and . (marked lines) may be used in place of a file name. The other parameters are optional and may appear in any order. Carriage control is added to skip to a new page every sixty lines unless CC is specified or the file's record length is 121 or 133.

Form 2 of the Editor PRINT command displays on the workstation the first *m* characters of the *n* consecutive lines beginning with the current line. Defaults are n=1 and m=record length.

## Parameters (Form 1):

filename — The name of the file to be printed. Under the Editor, the special name *CUR indicates the current contents of the file being edited, and the special name . indicates marked lines.

loc — The name of the printer where the file is to be printed. This may be an actual printer name or a printer location. It is 1 to 8 characters long. Some documentation refers to this as a "route name" or "routing name". The names are assigned by your system administrator.

If you do not specify a printer name, a default name is used. If you have used the command "ROUTE printername" previously in this MUSIC session, that name is the default. Otherwise, the name defined by ROUTE(name) in your User Profile (the PROFILE command) is used, if any. Otherwise, a default name based on your workstation location may be used. If none of the above cases apply, the name SYSTEM is used.

The following names are valid:

SYSTEM — Sends the output to the standard system printer.

MUSIC — Sends the output to the MUSIC Output Queue (the OUTPUT Facility).

DUMMY — Discards the output. Nothing is printed.

rscsname — The name of an RSCS printer. For example: R(PRINTER3) means the output is sent to RSCS and queued for printing on linkid PRINTER3.

prtname — The name of a MUSIC-controlled ASCII or 3270 printer as defined by your installation. Consult your installation for a list of valid names.

PC1 — A printer name such as PC1 may be defined by your installation. It prints the file on your PC printer (using DOS device LPT1), provided your PC is connected to MUSIC via NET3270 or PCWS. If the connection does not support PC printing, the data is sent to the MUSIC Output Queue (the OUTPUT Facility). Similarly, PC2 uses device LPT2 and PC3 uses device LPT3. When printing to a PC printer, some PRINT parameters such as COPIES, FORMS and PAGELEN may be ignored.

n — The number of copies that should be printed. The default is 1 copy.

x — A one to 8 character string indicating which forms are to be used when printing the file.

m — The number of lines per page. The default is 60 lines. This parameter is used only when the NOCC parameter is in effect.

CC — If specified, it indicates that the file to be printed already contains a carriage control character on the first character of each line in the file. The PRINT command attempts to honour these control characters.

NOCC — Indicates that the file does not contain printer control characters. The file is printed single spaced, with a skip to a new page after each m lines of output. NOCC is the default, except when the record length of the file is 121 or 133, in which case CC is assumed.

## Parameters (Form 2):

n — This specifies the number of consecutive lines that are to be printed. If * is specified, the rest of the file starting from the current line will be printed. If *n* is not specified, it is assumed to be 1.

m — This specifies the number of characters that are to be printed from each line. If not specified, the whole line will be printed.

**Examples:**

| | |
|---|---|
| `P` | Displays the current line. |
| `P5` | Displays 5 lines. |
| `PRINT * 72` | This prints the first 72 characters of the rest of the lines in the file, beginning with the current line. |
| `PRINT PGM R(PRTA)` | This prints the file PGM on the line printer called "PRTA". |

See also LIST, NUM, and WINDOW commands.

**PROMPT [x]**
**PR**

The character specified is used as a prompt character for workstation dialogue, and is displayed each time the Editor expects the user to type a line. This feature is particularly useful for TTY terminals where the keyboard is always unlocked. If *x* is not specified, Editor prompting is terminated if prompts were being issued, or, if prompts were not being issued, prompting is begun using a question mark character (?). At the start of the Editor session, a question mark (?) is automatically used as the prompt character for teletype (TTY) terminals.

**Spacing:** Blanks between the command and the character are ignored.

**PURGE name**
**PUR**

This command is used to permanently remove (delete) a file from the Save Library, as in the MUSIC command /PURGE. Alias: ERASE.

To delete the Input File, specify file name /INPUT. To delete the Holding File, specify file name /HOLD.

**Spacing:** The file name must be separated from the command by at least one blank.

```
QQUIT <n>
QQ
```

The QQUIT command immediately terminates the Editor session, without any messages or prompting. The file is not stored.

See QUIT for details and description of *n* parameter.

```
QUIT
Q
```

The QUIT command terminates the Editor session, without performing any save operation.

If you have made changes to the file but have not issued a FILE command to make the changes permanent, you will be prompted for permission to end the edit session. Enter YES or Y to end the session, or NO or N to cancel the QUIT operation and continue editing.

The END, QUIT, and QQUIT commands terminate the edit session without writing the editor's copy of the file to disk. An optional job return code *n* (a number 0 or higher) can be specified on each of these commands. If the return code parameter is omitted, the code specified on the last SETRC command is used, or 0 if no SETRC was done. Please refer to the SETRC command for additional notes.

```
RENAME oldname newname
REN
```

The RENAME command changes the name of an existing file in the Save Library. It performs the same function as the MUSIC command RENAME. The names can include an * (asterisk) to indicate groups of files.

Most attributes of the file, such as public/private, tag, and date last referenced, are unchanged by the command. The only exception is the date last written, which is set to the current date.

```
REPEAT  n
```

This command specifies the number of times the following BLANK or OVERLAY command is to be performed on successive lines.

To cancel the effect of a REPEAT command before BLANK or OVERLAY is used, enter the command REPEAT 1.

**Pointer:** Set to the last line changed.

**Spacing:** The number of times to repeat may be anywhere in the command line after the keyword.

**Example:**
```
Before:   * SAMPLE LINE       <--Pointer
          * ANOTHER LINE
Command:  REPEAT 2
Command:  OVERLAY C
After:    C SAMPLE LINE
          C ANOTHER LINE      <--Pointer
```

```
REPLACE  [string]
R
```

This command replaces the current line with *string*. You can replace the current line with a completely blank one by typing REPLACE without specifying a string.

**Spacing:** One blank must appear between the keyword and the first character of *string*. All succeeding characters are part of *string*.

**Example:**
```
Before:   SAMPLE LINE
Command:  R NEW STUFF
After:    NEW STUFF
```

```
REXX ON
REX  OFF
```

The REXX command enables or disables use of Rexx procedures from within the Editor. Rexx procedures are also called Editor macros. Refer to the section "Editor Macro Facility" for more information.

The REXX ON command must be used before Editor Rexx macros can be used. REXX ON could be contained in your private EDITOR file. Also, a large user region must be defined in order to use Rexx procedures: e.g. /SYS REGION=256. If the user region is not large enough to accommodate the Rexx processor and work area, the REXX ON command is ignored.

```
REX ON
```

This command must be done at least once in the edit session, before any Rexx procedures can be executed.

**Messages:**

`INVALID COMMAND OR REXX PROCEDURE NOT FOUND`
This message, issued when you enter an Editor command, means that Rexx procedures are enabled,

Rexx has been loaded, and the command you entered was not an Editor command, but Rexx could not find any procedure for that command name. (If REXX OFF is in effect, the message `INVALID COMMAND` is issued instead.)

`UNABLE TO LOAD REXX - COMMAND NOT DONE`
This message could be issued when the Editor tries to pass the command you entered to Rexx, since the command is not recognized as an Editor command. However, the Editor could not load the Rexx processor. The Rexx procedure you entered has not been done.

**RIGHT [n]**
**RI**

The command RIGHT shift the window display right a specified number of columns. These are equivalent to the command WINDOW RIGHT n. If *n* is omitted, the window is shifted the maximum number of columns possible.

See also LEFT, WINDOW and ZONE.

**RUN [name]**
**RU**

Usually the RUN command is used without a *name* parameter. When a name is specified, the RUN command has the same effect as the Editor EXEC command. If desired, column number limits and file attributes (PRIV, PUBL etc.) can be specified before and after *name* respectively (as on the FILE command).

When used without a *name* parameter, RUN saves the changed file in place of the Input file and then automatically requests MUSIC to execute the program from the Input file in a way equivalent to using the MUSIC command /RUN. This command is particularly useful when you want to make temporary changes to a file and do not wish to modify the original.

If this command is used in the User Data Set version of the Editor, either the Input file (if *name* is not specified) or the named file (if *name* is specified) will be replaced by the temporary updated version of the UDS file being edited. If the replacement is successful, a request to execute the program in the Input file or the named file will then be made to MUSIC. The original UDS file is not updated, and the edit is terminated.

**SAVE [name]**
**SA [(u) ]**
**SV**

This command causes the current temporary copy of the file to replace the specified file, without terminating the Editor. This is similar to the FILE command except that the edit session will not be terminated. The line

pointer is not changed. Therefore, the user may continue to issue additional edit commands on the same file.

**Parameters:**

name — If a file name is not specified on the command, the name on the original EDIT command or on the last NAME command will be used, or, in the case of editing a User Data Set (UDS) file, the UDS file being edited is replaced.

u — A unit number in parentheses may be specified in place of a file name, causing output to be done to that unit instead of to a file. For example, SAVE (3). The unit number must be 3, 7 or 10. The unit is not rewound before output is begun.

For a description of the various other parameters which can be used on the SAVE command, refer to the FILE command, which takes the same parameters. These include starting and ending column numbers and file attributes (PRIV, PUBL, etc.)

**Examples:**

| | |
|---|---|
| `SAVE NEWFIL` | It writes the updated version of file to a file named NEWFIL. The original file is not updated. |
| `SAVE 72,(3)` | It writes columns 1 to 72 of each line in the updated version of file to the UDS defined in unit 3. The original file is not updated. |

See also FILE and STORE.

**SCAN /string/[S]**
**SC**

This command searches the entire file being edited for all occurrences of *string*. Each line containing the string is displayed. The string must not be null, and the final delimiter (/) may be omitted if the S parameter is not given. This command is equivalent to a TOP followed by a "CHANGE /string/string/*,v."

If the parameter S is specified, the Editor will cause the screen to be displayed for each line found. On a 3270-type workstations, the ENTER key must be pressed between screens. The PA1 key can be pressed to stop the scan.

**Pointer:** Set to the bottom of the file.

**SCREEN [n]**
**SCR**

This command is useful for workstations that do **not** have full screen editing capabilities. It is used to turn on *screen* mode. The parameter *n* specifies the approximate number of screen lines to be used for displaying the current line and the lines above and below it. It must have a value from 7 to 17. At the start of the Editor session, *n* has a value of 15. If *n* is omitted, the number of lines is not changed. Refer to the section "Screen

Mode" for more details.

On a 3270-type workstation, full-screen (FS) mode is normally in effect, and overrides screen mode. To get screen mode, enter the commands NOFS and SCREEN, then enter a blank line.

```
SEARCH  [string]
S
```

The SEARCH command combines the effect of a TOP command followed by a LOCATE. It searches the entire file for the first line containing *string.* If in VERIFY status, the found line is displayed at the work-station. If *string* does not exist in the file, the message *EOF is displayed.

If *string* is not specified, the same string as specified in the last used FIND, LOCATE, UFIND, ULOCATE, HUNT, or SEARCH command is used.

**Pointer:** Set to the line containing *string,* or past the end of the file.

**Spacing:** One blank should be present between the keyword or abbreviation and the first character of *string.* All other blanks are part of *string,* which extends to the last non-blank in the command line.

```
SEARCHL  [n,](logical expression)
SL
```

The SEARCHL command is exactly equivalent to a TOP command followed by the corresponding LOCATEL command. Refer to the description of the LOCATEL command.

```
SEQ [xxx] [m] [n] [COL=k] [LEN=l] [LINES=s]
```

This command puts sequence numbers and an optional identification field into each line of the file. *xxx* is an optional identification field, up to 8 characters long, not all digits. *m* is the increment value, up to 8 digits. If *m* is omitted, a default value of 10 is used. *n* is the starting value for the sequence numbers, up to 8 digits. If *n* is omitted, the starting value will be the same as the increment.

COL=k and LEN=l may be used to specify the starting column *k* and the length *l* (1 to 8) of the sequence number field. Defaults are COL=73 and LEN=8.

If the parameter LINES=s is used, where *s* is a number, only *s* lines are changed, starting with the current line. The last line sequenced becomes the new current line. Note that if LINES=s is omitted, the entire file is sequenced.

**Pointer:** Set to the first line of the file (unless LINES=s is used).

**Spacing:** One blank may optionally appear between the command and the first parameter. The parameters are separated by blanks or commas.

**Examples:**

| Command | Resulting sequence numbers |
|---|---|
| `SEQ PGMA,1` | `PGMA0001,PGMA0002,PGMA0003 ...` |
| `SEQ 10,200` | `00000200,00000210,00000220 ...` |
| `SEQ ASM` | `ASM00010,ASM00020,ASM00030 ...` |
| `SEQ` | `00000010,00000020,00000030 ...` |

The following example shows how a series of 5 similar data lines, differing only by a sequence number field, may be inserted into a file:

```
Before:   xxxxxxxx
          yyyyyyyy       <--- current line
          zzzzzzzz
Command:  i name=abnnn
          dup 4
          up 4
          seq 1 1 col=8 len=3 lines=5
After:    xxxxxxxx
          yyyyyyyy
          name=ab001
          name=ab002
          name=ab003
          name=ab004
          name=ab005     <--- new current line
          zzzzzzzz
```

**SET xxxx**

The SET macro offers compatibility with the CMS editor: "SET xxxx" usually issues the command xxxx. For example,

```
SET NUM ON
```

As a special case, SET PFn xxx issues DEFINE PFn xxx.

**SETRC n**

The numeric value *n* (0 or more) specified on the SETRC command will be used as the job return code (exit code) for the edit, when the editor terminates. This includes all normal terminations, such as QUIT, QQUIT, END, OFF, FILE, etc. It does not include system aborts or /CAN used in attention mode.

If several SETRC commands are used, the last one is honoured. If no parameter is used on SETRC, the command is ignored. A return code specified on the QUIT, END or QQUIT command overrides any specified on SETRC.

If no return code is specified by SETRC, QUIT, END, or QQUIT, the edit return code is normally zero. However, a non-zero return code can result from an error during editor start-up (insufficient memory, file to be edited cannot be read, etc.) The return code in these cases is a small positive value. Also, some types of system errors can result in large return code values. Therefore, to avoid conflicts, applications should limit themselves to values for SETRC and QUIT/QQUIT/END of 0 or 200 thru 999.

Type "HELP RETURN" during an edit session to see a list of editor return codes. An editor macro can set a return code by the Rexx "EXIT n" statement. These do not affect the editor job return code.

**SETV xxx yyy**

The SETV command is mainly used when writing editor macros with REXX. It assigns a character string yyy to a name xxx. xxx is 1 to 8 characters. In a macro, you can retrieve the string yyy by "EXTRACT /$xxx/". This allows macros to remember things between calls. Omitting yyy undefines xxx, and the extract returns a null string. SETV definitions are local to the editor session in which they are made.

```
SETV ABCD This is the value
...
"EXTRACT /$ABCD/"     <-- sets abcd to "This is the value"
```

**SHIFT < n**
**> n**
**l**
**r**

The SHIFT macro shifts marked lines left (<) or right (>) by a specified number of columns (n). Text which is shifted left beyond column 1, or right beyond column n (where n is defined by the command ZONE n), is lost. Text outside the zone (columns 1 to n) is not affected.

See also MARK, ZONE, and PREFIX.

**SHOW PFn**
**SH PF**
**X**
**filename n1 n2**

The SHOW command displays the command string currently defined for a specific program function key (F1

to F24), or for the X command. Also, definitions of all 24 function keys may be requested. To get the definition of Xn (where n is 1 to 24), use "SHOW PFn".

The NOSHOW command removes the text displayed at the bottom of the screen by a previous "SHOW filename" command, and enlarges the FS screen back to its original size.

An alternate form of the SHOW command can display up to 12 lines from a specified file at the bottom of the screen in full-screen (FS) mode. The display will remain there until removed by a NOSHOW command or replaced by another SHOW command. The number of screen lines available for full-screen editing (i.e. the "n" in the command "FS n") is reduced by the number of lines displayed by the SHOW command. For example, this form of the SHOW command could be used to permanently display function key definitions or other helpful information.

Text for SHOW filename can contain 3270 Start Field orders (hex character 1D, followed by the field attribute byte). They can be used to highlight parts of the show text, or display it in a different colour. Attribute characters are D (unprotected low - green), H (unprotected high - red), U (protected low - blue, the default), and Y (protected high - white). The 1D character can be entered by using the editor XIN command. Each 2-character Start Field order occupies one blank position on the screen.

Within SHOW text, hex character 01 starts highlighting, and hex character 02 ends highlighting. This is an alternative to using Start Field (hex 1D). 01 and 02 are not followed by a field attribute character.

**Parameters:**

| | |
|---|---|
| PFn | (where n is 1 to 24) the Program Function key whose definition is to be displayed. |
| PF | Requests that the definitions of all function keys be displayed. |
| X | Requests that the definition of the X command be displayed. |
| filename | The name of a file whose contents is to be displayed at the bottom of the screen when in FS mode. |
| n1 | Starting line number in the file. The display starts with this line. Default for n1 is 1. |
| n2 | Ending line number in the file. The maximum number of lines that can be displayed is 12. The default for n2 is n1+11. |

**Examples:**

```
SHOW PF8            - Displays the current definition of F8.
SH PF               - Displays the definitions of F1 through F24.
SHOW INFOTEXT       - Displays contents of INFOFILE (max of 12 lines).
SHOW INFOTEXT 11 15 - Displays lines 11 through 15 of INFOFILE.
```

**SIZE**
**SI**

The SIZE command causes the Editor to display the total number of lines in the file. The line pointer is not changed.

```
SORT * parameters
     . parameters
     * filename2 parameters
     . filename2 parameters
     filename1 filename2 parameters
```

The SORT editor macro uses the MUSIC SORT command to sort the entire file you are editing ("sort *") or part of the file that you have MARKed ("sort ."). The sorted records replace the original records, unless you specify a target file name as the second parameter.

Parameters can be specified, if needed. They are the same as on the MUSIC SORT command.

Possible parameters are:

| | |
|---|---|
| m-n | Starting (m) and ending (n) column of the sort key. |
| -d | Sort in descending order. |
| -r | Replace the target file (filename2) without prompting. |
| -deldups | Delete output records which have the same key field as the previous output record (i.e. delete duplicates). |
| -xx | Sort control field type: -CH (character or binary), -FI (fixed-point), -FL (normalized floating-point), -ZD (zoned decimal), -PD (packed decimal), -DA (7-character date e.g. "02JAN89"). |

Default is to sort in ascending order, using the entire record (or the first 256 characters if the records are longer than 256) as the sort key.

When the first parameter is * or . , the records to be sorted are stored to file @SORT.TMP, which is then used as filename1 on a MUSIC SORT command. The resulting sorted file is merged back into the current file, replacing the original lines. The UNSORT macro uses the contents of file @SORT.TMP to undo the sort.

```
SPACE
SPA
```

The SPACE command displays the values from the UCR (User Control Record) for the user's userid. These are the user's file space limits and the total space currently used.

The unit is 1K = 1024 bytes. This information is especially useful for userids with limited file space.

```
SPELL
```

This macro invokes the SPELL program to spell check the current file. After you are finished with the SPELL program, you are returned to the original edit session.

*Note:* Invoking SPELL from the Editor instead of *Go mode means that no exception list is kept for your dictionary.

```
SPLIT  /string/[Cn][ICm]
SP
```

This command causes the current line to be searched for *string* and if it is found, the current line is broken into two lines, the first consisting of columns 1 through the last column before *string* and the second extending from the beginning of *string* to the end of the line. The C*n* parameter specifies the column number in which the search for *string* is to begin. The second line will be adjusted to start in column 1 unless IC*m* is used to specify an indentation column number. The first line becomes the new current line.

In 3270 full-screen mode only, the screen cursor may be used to indicate the point at which the line is to be split. In this case, the SPLIT command is used without any parameters. The character at which the cursor is placed becomes the first character of the second line. The cursor is left at the end of the first line. F2 has the default definition of SPLIT.

**Pointer:** Set to the first of the two lines.

**Spacing:** One blank may optionally be present between the command name or abbreviation and the first slash (/). The second string delimiter (/) may be omitted if Cn and ICm are not used and the string does not end in a blank.

**Example:**

```
Before:   SAMPLE LINE          <--Pointer
Command:  SP /LI/
After:    SAMPLE               <--Pointer
          LINE
```

```
STORE [n] [m] filename [attributes] [APPEND]
STO
```

The STORE command writes the group of marked lines to an external file. The edit continues. The MARK command (or function key operation) must be used to mark one or more lines before the STORE command can be used.

If the optional APPEND keyword (abbreviation: APP or A) is used following the file name, the marked lines are added to the end of the file, if the file already exists. A new file is created if one does not exist.

*Note:* When lines are appended to an existing file, unused space is not released at the end of the file. This is intentional, in order to prevent an excessive number of extents being created when several stores are done to the same file. Unused space can be freed later by editing and filing (FILE command) the file, or by running the following job:

```
/FILE 1 NAME(filename) OLD
/LOAD IEFBR
```

The following parameters are available:

| | |
|---|---|
| n | (optional) starting column number. |
| m | (optional) ending column number. |
| filename | the name of the target file. |
| attributes | (optional) file attributes: PUBL, PRIV, SHR, COM, XO, CNT. When appending to an existing file, the *attributes* are ignored. |

**Example:**

| | |
|---|---|
| `STORE MYFILE` | Writes marked lines to file MYFILE. (If the file already exists you are asked if you wish to replace it.) |
| `STO 11 20 MYFILE PUBL` | Writes columns 11-20 of marked lines to file MYFILE, and creates the file as public. |

**SUBMIT fn1 [fn2] [fn3] ... [kw1(value1)] [kw2(value2)] ...**
**SU**

This command submits one or more files as a job to MUSIC batch, or to other batch processors accessible at your installation.

*fn1, fn2, fn3,* etc. specifies the names of one or more files to be submitted. The special name of *CUR can be used to indicate the current contents of the file being edited. The files are concatenated in the order specified to form a single job.

*kw1, kw2,* etc. are keyword parameters whose values *value1, value2,* etc. are substituted into the control statements that are submitted with the job. These parameters are dependant on the particular processor to which the job is submitted, and are usually used to specify items such as time and page limits, passwords, output destinations, etc. The special keyword parameter TO(*processor*) can be used to specify the name of the processor (system) to which the job is to be submitted. Consult your installation for a list of valid processors. If any keyword parameters are specified, they must follow the file names specifications mentioned above.

Refer to the description of "Submitting Jobs to MUSIC Batch and Other Operating Systems" in *Chapter 3. Using Batch* of this manual for full details.

**SUBSET n**

The SUBSET command specifies which subset of the Editor commands is to be allowed during this edit session, or displays the number of the command subset currently in effect.

Parameter *n* is the number of the command subset which is to apply to the remainder of this edit session. The possible subset numbers are:

0 The full set of Editor commands. This is the default.

1 Subset used by the MUSCOM facility.

Note that once in this subset, you cannot get out of it!

2 Commands for read-only edit (BROWSE). No commands are allowed that would change the file being viewed. Screen text in 3270 full-screen mode is protected. SUBSET 2 is the default for the BROWSE command. You can return to the full command subset by entering SUBSET 0. However, logging (for edit restart) is not started if you used the BROWSE command or the RO option on the EDIT command.

If *n* is not specified, the number of the current command subset is displayed.

**Examples:**

```
SUBSET      Displays the current command subset number.
SUBSET2     Puts the Editor into read-only (BROWSE) mode.
SUBSET0     Returns to the full set of commands.
```

**Messages:**

`COMMAND NOT ALLOWED`
Means that the command you entered is not allowed in the current subset.

`OPERANDS ARE NOT ALLOWED ON THIS COMMAND`
Means that you may not use any parameters on this command in the current command subset. The command may be allowed without parameters.

**TABIN [c1] [c2] [c3] [c4] [c5]...**
**TABI**
**TABSET**
**TAB**

This command provides within the Editor the function of the MUSIC /TABIN command. The command is used to specify to the system the user's input tab settings. When the user then transmits a tab character, the system treats the input line as if the user had spaced to the next tab position. Workstation tabs need not be physically set to correspond with the columns specified in the TABIN command, although it is desirable to do so. (Note that the TABIN command may be used even for a workstation that does not have physical

tabs.) If this command is not used, tab settings remain as specified prior to the Editor session. When the Editor TABIN command is used, tab settings remain in effect after the Editor terminates. See /TABIN in *Appendix C. Seldom Used MUSIC Commands*. When TEXT SCRIPT is in effect tab characters are not expanded. Refer to the TEXT command.

**Spacing:** Column numbers specifying tab positions must be separated from each other by commas or blanks.

```
TABOUT  [c1] [c2] [c3] [c4] [c5]...
TABO
```

This command provides the function of the MUSIC /TABOUT command when you are using the Editor. The command is used to specify to the system the user's output horizontal tab settings. The system will then automatically take advantage of these tabs whenever possible to save typing time. The command permits the setting of up to eleven (11) output tab positions. These positions should be chosen to optimize printout speed, depending on the format of the output data. The first column is called tab position 1. Invalid tab numbers will be ignored. Tab settings remain in effect after the Editor terminates.

This command should not be used if the workstation is not equipped with horizontal tab capability.

**Spacing:** Column numbers specifying tab positions must be separated from each other by commas or blanks.

```
TAG  [string]
```

The TAG command displays or sets the tag string associated with the file being edited. The tag is up to 64 characters long, and is stored with the file when a SAVE, FILE, or EXEC command is used. Often the tag is used as a one-line description of the file's contents.

If *string* is specified on the TAG command, it replaces the current tag. To remove the tag, use the command TAG ' '. If no parameter is present, the current tag (if any) is displayed.

```
TEXT  [LC      ]
TEX   [UC      ]
      [SCRIPT  ]
      [NOSCRIPT]
```

The TEXT command is used to specify whether lower case input is to be translated to upper case or not. The command remains in effect until the end of the edit session, or until another TEXT command is used. When TEXT is entered without a parameter, the current setting is displayed.

**Parameters:**

| | |
|---|---|
| UC | specifies that lower case input is to be translated into upper case automatically. This is the default when the EDIT command starts the editor with a file that contains all upper case characters. |
| LC | specifies that lower case input is to be left as is. This is the default for new files and when starting the Editor with files that already contain lower case characters. |
| SCRIPT | indicates that lower case input is to be left as is, and that input tab characters are not to be translated to the appropriate number of blanks. Tab characters in the file are displayed as "#" characters. TEXT SCRIPT is the default if /TEDIT was used to start the edit. |
| NOSCRIPT | removes the effect of a previous TEXT SCRIPT command, except that the UC/LC setting is not changed. |

See also the CASE command.

```
TIME
TI
```

The TIME and USERS commands cause a line of status information to be displayed. The information includes time of day (in 24-hour clock), date, the amount of processing time (in service units) used since the start of the edit, and the number of MUSIC users on the system.

```
TOLC  [n]
```

This command changes characters to lower case in a line or group of lines, starting with the current line. The last line converted becomes the new current line. Only alphabetic letters (A-Z) in the *zone* portion (as defined by the ZONE command) of each line are changed.

*n* is the number of lines (1 or more). If *n* is omitted, only the current line is changed. The dot form of this command (TOLC.) can be used to change a marked group of lines.

```
TOP
T
```

The TOP command moves the pointer to the first line of the file.

**Pointer:** Set to the first line of the file. If the command INSERT, MERGE or INPUT is given immediately after the TOP command, the new lines are placed before the first line of the file. Also, a FIND, LOCATE or

SEARCH command used immediately after a TOP command will include the first line of the file in the search. At the start of the Editor session, the pointer is at the top of the file.

```
TOUC  [n]
```

This command changes characters to upper case in a line or group of lines, starting with the current line. The last line converted becomes the new current line. Only alphabetic letters (a-z) in the *zone* portion (as defined by the ZONE command) of each line are changed.

*n* is the number of lines (1 or more). If *n* is omitted, only the current line is changed. The dot form of this command (TOUC.) can be used to change a marked group of lines.

```
TRAN
```

The TRAN command restores the output character translation that was suppressed by a NOTRAN command.

```
TREE
```

This macro shows a graphical display of your directories. Same as the MUSIC command TREE.

```
UFIND  [string]
UF
```

The UFIND command searches the file, starting with the line preceding the current line, for a line beginning with *string*. The search works towards the beginning (top) of the file until the first match, or until the pointer is at the top of the file. In the latter case, the message `TEXT NOT FOUND` will result which means that *string* cannot be located between the line preceding the current line and the first line of the file. If VERIFY status is in effect, the line found is displayed at the workstation.

If *string* is not specified, the same string as specified in the last used FIND, LOCATE, UFIND, ULOCATE, HUNT, or SEARCH command is used.

The command can also be specified as UPFIND or UPF.

**Spacing:** One blank should be present between the keyword or abbreviation and *string*. It is ignored if present. All other blanks are considered to be part of *string*, which extends to and includes the rightmost

non-blank in the command line.

**Pointer:** Set to the found line, or to the top of the file if no line is found which begins with *string.*

**Example:**

```
Before:   SAMPLE LINE
          ANOTHER LINE
          YET ANOTHER          <--Pointer
Command:  UF SA
After:    SAMPLE LINE          <--Pointer
          ANOTHER LINE
          YET ANOTHER
```

```
ULOCATE  [string]
UL
```

ULOCATE is used to find the first line before the current line which contains *string* anywhere in the line and, if in VERIFY mode, to display the line on the workstation. The search begins with the line preceding the current line and works towards the beginning of the file. If *string* does not exist in the file between the line preceding the current line and the first line of the file, the message `TEXT NOT FOUND` is displayed. The operation of ULOCATE may be affected by the ZONE setting (refer to the description of the ZONE command).

If *string* is not specified, the same string as specified in the last used FIND, LOCATE, UFIND, ULOCATE, HUNT, or SEARCH command is used.

The command can also be specified as UPLOCATE or UPL.

**Pointer:** Set to the line in which the *string* is found, or to the top of the file, if it is not found.

**Spacing:** One blank should be present between the keyword or abbreviation and *string.* It is ignored if present. All other blanks are part of *string,* which extends as far as the rightmost non-blank in the command line.

**Example:**

```
Before:   SAMPLE LINE
          ANOTHER LINE
          YET MORE             <--Pointer
Command:  UL ER
After:    SAMPLE LINE
          ANOTHER LINE         <--Pointer
          YET MORE
```

```
UNDELETE
UNDEL
```

This command restores the line deleted by the previous DELETE command, provided it was a simple delete of 1 line. The deleted line is inserted before the current line.

```
UNFF
```

This macro undoes the immediately preceding FF macro or PREFIX command.

```
UNFORMAT
```

This macro is used to undo the effect of the FORMAT macro. It restores the formatted file back to its unformatted form that was previous to using the FORMAT macro.

```
UNMARK
UNMA
```

This command unmarks any lines that were previously marked by the MARK command. Refer to the topic "Marking a Group of Lines" earlier in this chapter.

```
UNSORT *
       .
```

The UNSORT editor macro can be used to undo the effects of the immediately preceding SORT macro.

You should specify the same parameter (* or .) as you used on the sort command.

UNSORT uses the contents of file @SORT.TMP, which is created by the SORT command.

```
UP  [n]
U
```

This command moves the pointer *n* lines toward the beginning of the file. If *n* is omitted, 1 is assumed.

If the number of lines specified is large enough to cause the line pointer to be moved beyond the first line of the file, the UP command has the same effect as a TOP command. For example, an immediately following INSERT command would insert a line before the first line.

```
UPPAGE
UPP
```

This command is used only in 3270 full-screen mode. It shifts the screen display one screen (page) towards the beginning of the file. Refer to the section on full-screen mode for more details. UPPAGE corresponds to F7 by default.

```
UPWINDOW  [n]
UPW
```

This command applies only to 3270 full-screen mode. It shifts the screen display (window) towards the beginning of the file. The parameter *n* is the number of lines by which the window is to be shifted. If *n* is omitted, 6 lines is assumed.

```
USERS
US
```

The TIME and USERS commands causes a line of status information to be displayed. The information includes time of day (in 24-hour clock), date, the amount of processing time (in service units) used since the start of the edit, and the number of MUSIC users on the system.

```
VERIFY  [n  ]
V       [ON ]
        [OFF]
```

VERIFY specifies that found or changed lines are to be displayed at the workstation after the following commands: ADD, BLANK, CHANGE, CHANGEL, DELETE, DELETEL, FIND, HUNT, LAST, LOCATE, LOCATEL, NEXT, OVERLAY, SEARCH, SEARCHL, SPLIT, UP. The command VERIFY OFF turns off verify mode (this is equivalent to the BRIEF command). The command VERIFY ON turns on verify mode (same effect as VERIFY without an parameter). If a number *n* is specified, it defines the window as columns 1 to *n* (refer to the WINDOW command), meaning that only columns 1 to *n* of each line are displayed. If an asterisk (*) is specified for *n*, *n* will have a value which is the record length of the file being edited. If *n* is omitted, the window setting is not changed.

```
WINDOW  [n      ]
W       [m n    ]
        [OFF    ]
        [RIGHT k]
        [LEFT k ]
        [FLIP   ]
```

This command specifies the starting and ending columns to be displayed by the Editor whenever a line of the file is displayed. This is particularly useful when long records are being edited. The listing of an external file, by the LIST command, is not affected.

*m* is the starting column number. *n* is the ending column number, and may be specified by an asterisk (*), which means the record length of the file being edited. If only one number is specified, the number is assumed to be *n*, and *m* is assumed to have a value of 1.

If OFF is specified instead of column numbers, the Editor resets the *window* to be the whole line, as it is when the Editor starts.

The parameter RIGHT or LEFT may be used on the WINDOW command to shift the window towards the right or left hand side of the records being edited. The width of the window is not changed. The parameter *k* is the number of columns by which the window is to be shifted. If *k* is omitted, the window is shifted by the maximum amount. RIGHT and LEFT may be abbreviated R and L.

The FLIP parameter causes the window to be shifted to the extreme left or right, whichever is further from the existing window setting. FLIP can be abbreviated FL or F.

When a WINDOW command is used in 3270 full-screen mode, the zone is automatically set to be from column 1 to the end of the window (refer to the ZONE command).

If no parameters are specified, the current window setting is displayed.

The window setting can also be changed by specifying a number *n* on the VERIFY command. In this case, the window setting will become columns 1 to *n*.

When the WINDOW command is used, the portion of a line displayed by the PRINT command defaults to the window setting. The second parameter on the PRINT command may still be used to display more or fewer characters per line starting from column 1.

**Spacing:** The parameters must be separated from each other by commas or blanks. If OFF, RIGHT or LEFT is specified, it must be separated from the command by at least one blank.

See also LEFT and RIGHT.

```
X  [n]
```

The X command causes a predefined set of Editor commands to be executed. The commands are defined by the DEFINE command. For example, if "DEFINE X LOCATE ABC;INSERT **" is used, then entering the

command X causes the commands "LOCATE ABC" and "INSERT **" to be performed. An X command string may not contain an X command. The X command may be used on all types of workstations.

If parameter *n* is used (a number from 1 to 24), the command string defined for program function key *n* (PFn) is executed. In this way, function key operations can be defined and used on any type of workstation.

```
XIN
```

This command causes the Editor to go into hexadecimal input format. This form of input is discussed at the beginning of this section. The command AIN is used to return to the regular input format.

```
XL string
XUL string
XF string
XUF string
```

The XL macro locates a character string that you specify. It issues a LOCATE command for the specified character string. However, if the search is unsuccessful, XL does not alter the current line pointer. This has the effect of not altering the current screen when the specified string is not found. If the specified character string is not located, then the current line number is not changed, and a message is displayed indicating that the search failed.

All of the considerations applicable to the LOCATE command apply to the XL macro.

XUL This is the same as the XL macro except that ULOCATE command is used.

XF This is the same as the XL macro except that FIND command is used.

XUF This is the same as the XF macro except that UFIND command is used.

```
ZONE  [n]
Z     [FIXED]
```

The ZONE command is used to display the current zone setting, or to change the zone setting. The zone setting defines the ending column for CHANGE and ADD commands, and for commands which may scan for character strings (such as DELETE, FIND, HUNT, UFIND, ULOCATE, LOCATE, SCAN, SPLIT, and SEARCH), including logical expressions. Columns following the ZONE specification are not changed by CHANGE or ADD commands and are not examined when looking for a character string. ZONE does not affect BLANK, OVERLAY and PRINT commands.

When the SPLIT command is used to break a line into two parts, the second part extends only as far as the

zone setting. Characters following the zone remain on the first line.

*n* must be a number from 1 to the record length. * is equivalent to specifying the record length, which is the value assumed when the Editor begins execution. If no parameter is specified, the current zone specification is displayed.

The option FIXED (abbr. F) keeps the zone setting unchanged by later WINDOW, LEFT, RIGHT or FS commands (e.g. ZONE * F). The command ZONE F sets the "fixed zone" option without changing the current zone setting. See also the WINDOW command.

```
* comment
```

Any command line starting with an asterisk (*) is treated as a comment, and is ignored by the Editor.

Also, if an Editor command line has an asterisk in column 1, the line is not searched for command delimiter characters (normally semicolon, ";"). But if the * command is not the first one on the line, then delimiter characters are honoured. For example, TOP;*XXX;BOTTOM does a TOP then a BOTTOM, while *XXX;TOP is entirely a comment (no TOP is done).

```
=
=n
=n/string1/string2/[options]
=.
=label
```

This command, without any parameters, causes the Editor to display the line number of the current line. Lines of the file are numbered consecutively starting at 1. When you see the message *EOF (pointer beyond the end of the file), the number of lines in the file, plus 1, will be displayed.

If a line number *n* is specified with the command, the line pointer moves to the line numbered *n* and the line becomes the new current line. The line is also displayed. For example, =30 makes the line numbered 30 the new current line. *n* can be 0 to go to the top of the file or a number greater than the file size to go to the bottom of the file.

If "=." is specified the Editor moves to the first line in a marked group.

If "=label" is specified the Editor moves to the line that was previously labeled with the POINT command.

A change request, with or without options, as in the CHANGE command, can be specified following the line number. Refer to the description of the CHANGE command. For example, =30/ABC/XYZ/ causes the line pointer to move to line 30 and the string ABC is changed to XYZ. The changed line is displayed.

**Examples:**

```
=                   Displays the line number of the current line.
=300                This sets the line pointer to the line number 300.
=300/NEGATIVE/POSITIVE/

                    This sets the line pointer to the line number 300 and changes the
                    "NEGATIVE" to "POSITIVE" in that line.
=here               This goes to the line that was previously labeled "here" with the
                    POINT command.
```

# Advanced Features

The Editor has several advanced features that can be of great assistance in some cases. It is recommended that the user become familiar with the basic functions before studying these advanced features.

## Starting Column Suffix -- CN

A number of edit commands can be modified so that their effect will start at a specified column number. For example, you can use the command FC7 WRITE to do a find operation using column 7 rather than 1 -- a great help for FORTRAN programs!

The following is a list of the various Editor commands that take the column number modifier. The notation *Cn* is used where the *n* is the column number from 1 to 8192.

| | |
|---|---|
| FINDCn string (abbrev. FCn) | a FIND in column n instead of column 1 is performed. |
| UFINDCn string (abbrev. UFCn) | a UFIND in column n instead of column 1 is performed. |
| HUNTCn (abbrev. HCn) | a TOP followed by a FINDCn. |
| LOCATECn string (abbrev. LCn) | a LOCATE starting in column n is performed. |
| ULOCATECn string (abbrev. ULCn) | a ULOCATE starting in column n is performed. |
| SEARCHCn string (abbrev. SCn) | a TOP followed by a LOCATECn. |
| OVERLAYCn string (abbrev. OCn) | *string* is overlaid starting in column *n.* |
| BLANKCn string (abbrev. BLCn) | a BLANK operation starting in column *n* is undertaken. |
| INSERTCn string (abbrev. ICn) | a new line is created, composed of *n-1* blanks followed by *string.* |
| REPLACECn string (abbrev. RCn) | same as INSERTCn, except that new line replaces current line. |

For example, to place an X in column 72 of the current line, one might type OC72 X. To remove this X, one might then type BLC72 X. If it is desired to use a *flip* character with any of these modified commands, the flip character must be placed immediately after the last digit in the column number. (The flip character is described under the FLIP command.)

## Logical Commands

Several Editor commands have versions which permit the use of *logical expressions*. A logical expression is one or more delimited strings, each of which is considered to have a value of *true* or *false* depending on whether it appears or does not appear in a line. The entire expression must be enclosed in parentheses and may contain any number of nested parenthetical expressions. The string delimiter used is normally a slash (/), but may be any non-alphanumeric character not appearing in the string, except the *not* ¬ symbol. The following connective operators may be used:

| | |
|---|---|
| OR or \| | inclusive "or" |
| AND or & | "and" |
| XOR or X | exclusive "or" |

The following forms of the *not* modifier may be used:

NOT or .NOT or ¬

Any *not* operation is performed before any *and* and *or* operations at the same level of parentheses, but *and* and *or* operations at the same level are performed from left to right, and *and* does not necessarily have priority over *or*. For this reason, it is recommended that parentheses be used freely to define the order of evaluation of the logical expression. For example, use (/a/or(/b/and/c/)) rather than (/a/or/b/and/c/).

The manner in which the string is searched for may be varied by the use of a string modifier, which is a parenthetical series of parameters placed immediately after the second delimiter. The following parameters apply:

F string must start in the column defined by the Cn parameter, or in column 1 if a Cn parameter is not used in the modifier.

Cn specifies n as the column to begin the search for string.

The following are examples of logical expressions:

```
(/xxx/and/yyy/)

(/xxx/(fc25))

(/xxx/(f)and/yyy/and.not(/jjj/or/kkk/or(/e/and/t/)))

(/aaa/|¬/bbb/(c3)|¬(/rrr/&/ttt/(c16f)))

(not/abcdef/)
```

# Defining Function Keys and the X Command

Users editing in full-screen mode on 3270-type terminals may customize the operation of the function keys. This is done by the DEFINE Editor command, which defines a function key as equivalent to any string of Editor commands. Any function keys defined in this way override the standard default function key definitions.

*Note:* It is possible that your installation has changed the standard default function key assignments, in which case the function key definitions described above may not be accurate. You can check this by entering the command SHOW PF.

For terminals that do not support function keys, the DEFINE command can make the X command equivalent to a string of Editor commands. Then typing the command X causes the specified sequence of commands to be executed. Also, the Xn command (where *n* is a number from 1 to 24) can be used on any type of terminal to execute the command string defined for Fn.

Format:

```
DEFINE Fn  commands
       X
```

*n* is the program function key number (1 to 24). *commands* are a string of 1 or more Editor commands, separated by the command delimiter character (normally semicolon, ";"). The DEFINE command must be the only command on the input line. An X command string must not contain an X command.

If the command string is omitted, the function key or X command is made undefined.

The abbreviation DEF may be used for DEFINE.

## Creating your own Editor

To have certain definitions in effect whenever you use the Editor, create a file named "EDITOR" consisting of /INCLUDE *COM:EDITOR followed by the desired DEFINE commands (plus any other Editor commands).

As an example, suppose you wanted your delete character to be ">" rather than ":", F23 to insert 8 blank lines, F21 to be undefined, F13 and 14 to be MOVE., COPY. and F18 to be the SAVE command followed by SUBMIT *CUR. You could do this by creating a file named "EDITOR" containing the following lines:

```
/INCLUDE *COM:EDITOR
DELCHAR >
DEFINE F23 MINSERT 8
DEFINE F21
DEFINE F13 MOVE.
DEFINE F14 COPY.
DEFINE F18 SAVE;SUBMIT *CUR
SHOW MYPFK
TOP
```

The command SHOW MYPFK permanently displays the contents of file MYPFK at the bottom of the screen. This file could indicate your program function key definitions, as a reminder during the edit session.

The SHOW command is used to display the current definition of program function keys or the X command: SHOW PFn, SHOW PF, or SHOW X.

The ECHO command is specially designed to be used on function keys. Its main function is to cause a string of Editor commands(s) to be displayed in the command area of the screen when a certain function key is pressed. You can then make minor modifications to the command string and subsequently execute it by pressing the ENTER key. As an example, assume that you define the F10 key with the command

```
DEF F10 ECHO MERGE ABC 1 2;LOC XYZ
```

When the F10 key is pressed, the string

```
MERGE ABC 1 2;LOC XYZ
```

is displayed in the command area. You may then modify the line numbers of the MERGE command and press the ENTER key to execute the MERGE and LOCATE commands.

# Editing a Large File

The Editor has a work file large enough to edit a file of about 4500 80-byte records (or the equivalent). If you must edit a larger file, or if you get the message `WORK FILE TOO SMALL -- nnnn RECORDS LOST AT END` when you try to edit a file, execute the following job instead of using the EDIT command:

```
/PARM filename options
/FILE 1 UDS(uuuucccc) NREC(nnnnn) VOL(volume) NEW DELETE
/INCLUDE EDITOR
```

The information on the /PARM statement is the same as you would use on the EDIT command. The value of NREC should be large enough to make the work file (unit 1) be at least 2.1 times the size of the file to be edited. For example, to edit a file of 20000 80-byte records, NREC should be approximately

```
(2.1 * 20000 * 80) / 128 = 26250
```

See the MUSIC command called "BIGEDIT" for editing large files.

# Editing User Data Set Files

To edit a user data set (UDS) file, execute the following job, using an appropriate /FILE statement pointing to your UDS file:

```
/FILE 4 UDS(dsname) VOL(volume) OLD
/INCLUDE EDITOR
```

Note that the /FILE statement uses unit 4 to point to your UDS file. The OLD on the /FILE allows you to write back the updated version. Use SHR if you only want to look at the file and want the system to protect you from writing on it.

The above control lines can themselves be saved in a Save Library file, in which case you would only have to type the Save Library file name at *Go time to run the edit program on your data set.

The Editor cannot edit files with a record size (RSIZ or LRECL) which is greater than 512 bytes.

The UDS Editor has enough working space to edit files that hold up to 4500 80-byte records or the equivalent. To edit larger files you must provide a temporary UDS file on unit 1 as shown in the following example:

```
/FILE 4 UDS(dsname) VOL(volume) OLD
/FILE 1 UDS(uuuucccc) NREC(nnnnnn) VOL(volume) NEW DELETE
/INCLUDE EDITOR
```

The value of NREC should be large enough to make the temporary UDS file at least 2.1 times the size (in bytes) of the UDS being edited. The size of a file in bytes can be calculated by multiplying the record length (RSIZ) by the total number of records (NREC). The data set name for the temporary file (uuuucccc in the above example), must agree with the usual data set naming conventions for UDS files.

You can provide a /FILE statement for a unit 3 if you wish to write on another file using the unit number option on the Editor FILE command, or to access the contents of another file using the unit number option on the MERGE or LIST command.

# Additional Editor Command Input

Commands to be executed by the Editor come from three sources:

1. Commands in the input stream (unit 5), following the /INCLUDE EDITOR statement.

2. Commands (if any) on the same line as the EDIT command, separated from the filename and from each other by semicolons.

3. Normal Editor commands entered conversationally by the user.

Commands from these sources are processed in the order given above.

To supply input stream commands to the Editor, execute the following job instead of using the EDIT command:

```
/PARM filename options
/INCLUDE EDITOR
...Editor commands...
```

The lines following /INCLUDE EDITOR may be Editor commands, or /INCLUDE statements pointing to save files containing Editor commands. The Editor always starts off in command (Edit) mode (not Input mode) when processing these commands.

If you wish to have certain Editor commands always executed at the beginning of each edit, save (under your user code) a private file called EDITOR, consisting of /INCLUDE *COM:EDITOR followed by the desired commands. For example:

```
/INCLUDE *COM:EDITOR
USERS
FLAG SCRIPT
```

Then, whenever you use the EDIT command or /INCLUDE EDITOR (while on your user code), the commands USERS and FLAG SCRIPT will be performed at the beginning of the edit.

# Specifying Editor Comments (* Command)

Any command line starting with an asterisk (*) is treated as a comment, and is ignored by the Editor.

Also, if an Editor command line has an asterisk in column 1, the line is not searched for command delimiter characters (normally semicolon, (;)). But if the * command is not the first one on the line, then delimiter characters are honoured. For example, TOP;*XXX;BOTTOM does a TOP then a BOTTOM, while *XXX;TOP is entirely a comment (no TOP is done).

# Defining a TAB key

A function key can be defined to perform a tab operation for the Editor. For example, DEFINE F4 <TAB> allows you to advance the cursor to the next tab position, as defined by the TABIN command.

*Note:* A local tab key is more efficient, and should be used when available (3270 Entry Assist and PCWS).

## Using Full-Screen Mode Without Function Keys

If you wish to use full-screen mode with a 3270-type terminal which does not have any program function keys, use the command FS NOPFK. This causes the Editor to put the cursor in the command area whenever the screen is displayed, thus facilitating the entry of commands. To move the cursor to the current line, leave the command area null and press the ENTER key, or simply use the arrow keys.

In addition, the commands UPWINDOW, DOWNWINDOW, UPPAGE and DOWNPAGE may be used in place of function key operations.

## Simulating Additional Function Keys

It is possible to make use of all 24 function key operations even if your terminal only has keys for F1 to 12. The extra keys for F13 to 24 are simulated by placing the cursor into the left-hand margin of the screen, one column to the left of a screen line or the command area, and then pressing the corresponding function key (1 to 12). The Editor detects the special cursor position, and simulates the appropriate function key (it adds 12 to the number).

In this way, the LEFT ARROW (<--) key acts somewhat like a *shift* key for the function keys.

The special cursor position is screen column 4 (without line numbers) or column 7 (with line numbers) for a line of the file, and is column 9 for the command area line.

If the cursor is in the margin column when a real F13 to 24 is pressed, then the Editor will simulate the corresponding F1 to 12 key (it subtracts 12 from the number).

## Blank-Filled Screen Display

Normally, when the screen is displayed, blanks at the end of each field (i.e. each line of the screen) are replaced by null characters. This allows easy use of the INS MODE key for inserting characters in the middle of a line. When adding characters to the end of a line, be careful to use the space bar to enter blanks, rather than the arrow keys, since null characters are not transmitted to the Editor.

However, for some applications such as entering tables or diagrams, it is more convenient to have trailing blanks retained. This is requested by using the FILL command. If the INS MODE key must be used when FILL is in effect, first use the ERASE EOF key to remove trailing blanks from the field. The NOFILL command reverts to the normal method of display.

An alternate name for the FILL command is NONULLS; an alternate name for the NOFILL command is NULLS. The command NULLS FLIP reverses the NULLS setting.

FILL (or NONULLS) mode is indicated by the ")" character at the end of the tab line.

## Output Cursor Positioning

When the Editor displays the screen, it normally puts the cursor at the first position of the current line or the command area. However, commands such as CURSOR, LOCATE and SPLIT can cause the cursor to be displayed at a different position in the current line.

The CURSOR command controls output cursor placement:

```
CURSOR LOCATE
CU     NOLOCATE
       n
       n TEMP
       END
```

*n* is a number from 1 to 72. It is a column position number relative to the start of the screen field for the current line. 1 refers to the first character of the field, 2 to the second character, etc.

CURSOR LOCATE places the cursor at the start of the found string after any of the commands: LOCATE, UPLOCATE, FIND, UPFIND, SEARCH, HUNT. CURSOR NOLOCATE cancels this option. You can put a CURSOR LOCATE command into your private EDITOR file to make this option the default for all your edits. Abbreviations are LOC, NOLOC.

"CURSOR n" places the cursor at the specified column position whenever the cursor is not put out in the command area, provided a command such as LOCATE or SPLIT or CURSOR END or "CURSOR n TEMP" does not result in a different placement. This stays in effect for the remainder of the edit. To cancel the effect of this command, use CURSOR 1.

"CURSOR n TEMP" places the cursor at position n in the screen field for the current line, but only for the next screen display. Abbreviation T may be used for TEMP, as in CU 15 T. This command is intended mainly for function key definitions. For example, DEFINE F1 INSERT ABC---XYZ;CURSOR 4 TEMP.

CURSOR END is similar to "CURSOR n TEMP", except that the cursor is placed after the last non-blank character in the field.

## The S Parameter on the SCAN and CHANGE Commands

When the *S* parameter is used on the SCAN or CHANGE command while in full-screen mode, each found or changed line is displayed on a new screen along with the preceding and following lines, and `More...` appears in the bottom right corner of the screen. Press the ENTER key to advance to the next screen. Screen changes are not allowed.

**Example:**

```
SCAN/ABC/S
CHANGE/ABC/XYZ/*S
```

If you wish to terminate the SCAN or CHANGE command before all the screens have been displayed, press the PA1 key.

*Note:* You may find it more convenient to use the LOCATE function key (F9) rather than the SCAN command. Enter a LOCATE or SEARCH command to find the first occurrence of the string, then press F9 to find each subsequent occurrence.

## RIGHT and LEFT Parameters on the WINDOW Command

The parameter RIGHT or LEFT may be used on the WINDOW command to shift the window towards the right or left hand side of the records being edited. The width of the window is not changed. Optionally, the number of columns for the shift may be specified. Also, the FLIP parameter can be used to shift the window

to the extreme left or right, whichever is further from the existing window setting. These parameters may be used whether or not full-screen mode is in effect. The format of the commands is:

```
WINDOW RIGHT

WINDOW RIGHT n

WINDOW LEFT

WINDOW LEFT n

WINDOW FLIP
```

The parameter *n* is the number of columns by which the window is to be shifted. If *n* is omitted, the window is shifted by the maximum amount. RIGHT and LEFT may be abbreviated R and L. WINDOW FLIP shifts the window to the extreme opposite side.

When a WINDOW command is used in full-screen mode, the zone is automatically set to be from column 1 to the end of the window (refer to the ZONE command). So remember that if you have used a command such as ZONE 80, you must re-issue it after shifting the window left or right, otherwise the zone may be reset to 72. The ZONE setting affects which columns are changed by the CHANGE command.

Example: Assume the length of the records being edited is 80, and the current window is columns 1 to 72. Then "WINDOW RIGHT 2" results in a new window setting of 3, 74. WINDOW RIGHT or WINDOW FLIP results in a window setting of 9, 80.

## Hexadecimal Input and Output

When the Editor begins, it operates in the normal alphanumeric input and output mode. Input and output may, however, be performed in hexadecimal form by using the commands described below. (Hexadecimal format represents each storage byte as two hex *digits*. Each hex digit can have 16 possible values. These 16 possibilities are represented by the sequence 0-9, A, B, C, D, E, F.)

| | |
|---|---|
| HEX | Output in hexadecimal form only, 32 bytes per line. |
| ALPHA | Restoration of normal alphanumeric output format. |
| BOTH | Output in hexadecimal form with each alphanumeric character printed above the left-hand position of the corresponding hexadecimal representation. |
| XIN | Input in hexadecimal format (see below). |
| AIN | Restoration of normal alphanumeric input format. |
| NOTRAN | Normally all output from the Editor is translated to ensure that only printable characters are typed. NOTRAN suppresses this translation. |
| TRAN | Restoration of normal character translation. |

# Hexadecimal Input Format

During command mode with hexadecimal input format (XIN) in effect, each line typed by the user is considered to have two fields, separated by one or more blanks. The fields are interpreted as follows:

FIELD 1 Starts at the 1st character and extends to the first blank in the line (or to the logical end of the line if no blank is present). This field always specifies an edit command and is always interpreted as normal alphanumeric input.

FIELD 2 Starts after the first blank and extends to the logical end of the line. In this field any of the 256 EBCDIC characters can be specified by the corresponding hexadecimal value. Blanks are ignored and do not affect the interpretation of the line. Any non-blank character which is not a valid hexadecimal (that is, not one of 0-9, A-F) is taken as is, unless there is an error in the previous part of the line. Any characters, including blanks, may be specified literally by enclosing them in single quotes. To specify a literal single quote, two consecutive single quotes are typed.

In Input mode, the entire line is in FIELD 2 format.

*Notes:*

1. Although N6 and N 6 are equivalent in AIN mode, they are not equivalent in XIN mode.

   N6, N '6' and N F6 are all equivalent in XIN mode.

2. It is not necessary to close single quotes at the end of a line. End-of-line performs this function automatically.

3. The user using XIN mode must be careful not to code a delimiter in the middle of a string. For example, since a slash (/) is equivalent to hexadecimal 61, the command C /05A1/0461/ would be equivalent to C /05A1/04// which is not a valid command format.

4. The command delimiter character (normally semicolon) will not be recognized as a command delimiter if it is entered in its 2-character hexadecimal form.

# Changing Screen Colors

```
COLOR
COLOR fieldname=color fieldname=color...
COLOR Base
COLOR Defaults
```

(Alias: COLOUR). Abbreviations: B for BASE and D for DEFAULT.

The COLOR command redefines the color of various parts of the editor screen. This assumes that the terminal supports 3270 extended data streams (i.e. the Start Field Extended order). Not all terminals support all the possible colors.

The BASE parameter turns off use of 3270 extended attributes (SFE). It reverts to the basic 4 colors (using

SF). Previous extended color settings are still remembered, and are reinstated when a COLOR command without the BASE parameter is used (including a COLOR command with no parameters).

The DEFAULTS parameter turns off use of extended attributes (SFE) and also resets all the field colors to their defaults. This undoes any previous use of the COLOR command.

"fieldname" is the name of a field or group of fields on the editor screen. Only the first 3 characters are significant.

| | |
|---|---|
| MISCEL<br>IDLINE<br>ID<br>I<br>TAB<br>STATUS<br>ST | Title line, tab line, "Command", "Input Mode", status word ("Reading" etc.) (default: blue) |
| MSG | Message area (white) |
| CURLINE<br>CU<br>CURREC | Current line, unprotected (red) |
| BCURLINE<br>BCURREC | Current line, protected (as in BROWSE) (white) |
| FILEAREA<br>FI<br>F<br>RECORDS | Non-current lines, unprotected (green) |
| BFILEAREA<br>BF<br>BRECORDS | Non-current lines, protected (as in BROWSE) (blue) |
| TOFEOF<br>TO | Top-of-file and end-of-file lines (white) |
| MARGIN<br>MARK<br>NUM<br>PTR | Arrow pointer, protected line numbers, mark character (blue) |
| PREFIX<br>PR | Prefix area (green) |
| CMD<br>CM | Command area (green) |
| SHOW<br>SH | SHOW text, not highlighted (blue) |
| HISHOW<br>HSHOW | SHOW text, highlighted (white) |

"color" is a 3270 color name or number. The following colors can be used. Only the first 8 characters are significant in the color names.

| Color # | Color Name | Abbreviations or aliases | |
|---|---|---|---|
| 0 | DEFAULT | DEF DE D | <-- usually green |
| 240 | NEUTRAL | | <-- usually green or black |
| 241 | BLUE | BL B | |
| 242 | RED | RE R | |
| 243 | PINK | P | |
| 244 | GREEN | GR G | |
| 245 | TURQUOISE | TURQ T | |
| 246 | YELLOW | Y | |
| 247 | WHITE | WH W | |
| 248 | BLACK | | |
| 249 | DEEPBLUE | | |
| 250 | ORANGE | | |
| 251 | PURPLE | | |
| 252 | PALEGREEN | | |
| 253 | PALETURQUOISE | | |
| 254 | GRAY | GREY | |
| 255 | BRIGHTWHITE | HIWHITE HIGHWHITE HIWH HIW HWHITE HWH HW | |

**Example:**

```
pref on
color pre=y
```

If the SFE (Start Field Extended) 3270 order is not supported by the terminal, or if an unsupported color is specified, an FSIO error message may appear, and the editor may revert to the base colors (COLOR BASE). Most newer 3270s support SFE and color numbers 0 and 241 to 247. Versions of Net3270 dated Sept./90 or later support SFE and all color numbers. PCWS does not support SFE.

With Net3270, you can define the PC-PS/2 color to be displayed for a given 3270 color name. For example, to define 3270 yellow as PC-PS/2 brown, press Alt-O to invoke Net3270's options screen, and enter the command "color 3270-yellow brown". You can also use Net3270's NETCOLOR program to define the red, green and blue components of a PC-PS/2 VGA color such as brown. This lets you customize the exact appearance of each color.

# Editor Macro Facility in REXX

Editor macros can be written in the REXX language. The macros can issue Editor commands, issue MUSIC commands, extract information from the Editor using the EXTRACT command, call other macros, and use the MUSIO and PANEL commands available in REXX. The EDITOR/REXX interface must first be enabled via the REXX ON command (this is the default setting). Once enabled, any command that is not a valid Editor command is passed to REXX. REXX searches for the source of the macro in a PDS structure called MACLIB. If the REXX macro has the same name as an Editor command, the user can use a % symbol preceding the command to distinguish it from the Editor command. If it cannot find the macro, the command is executed as if it were a MUSIC command. This macro facility requires that the Editor be run in at least a 256K region, which is the default. Large macros may require more storage.

## Writing Editor Macros

The macros are written in the standard REXX language. Unlike normal REXX programs, the /LOAD REXX or /INC REXX statements that normally appear at the beginning should NOT be present. Also, unlike normal REXX programs, Editor macros cannot issue regular MUSIC commands. Valid commands are MUSIO, PANEL, EXTRACT, and any Editor commands, including any other macros. The parameter entered with the macro command is available through the regular REXX PARSE ARG statement. Editor command strings issued from the macros must contain only one Editor command, since the interface does not support commands separated by the Editor command delimiter. Multiple Editor commands can however be placed on the same line in the macro by using the REXX statement delimiter. Note that quotes and upper case should be used. For Example:

```
'TOP;L XXX;DEL 3'          is invalid.
```

however.....

```
'TOP'; 'L XXX'; 'DEL 3'    is valid.
```

## Return Codes

Most Editor commands, macros, and MUSIC commands set a *return code* to indicate the success or failure of the operation. Usually 0 indicates success and a non-zero value indicates an error or an unusual condition. After executing a command or macro, a REXX procedure can test the return code, which is in the special REXX variable RC. To see the various return code values set by Editor commands type "help return" in the Editor's command area.

## The EXTRACT Command

The EXTRACT command is used by macros to get information about the current edit session. It causes the Editor to set the requested REXX variables to the appropriate values. The format of the EXTRACT command is:

```
'EXTRACT /varnam1/varnam2/...../varnamx/'
```

This command instructs the Editor to set the variables *varnam1* through *varnamx* to the appropriate values.

The variables that can be set and the values that they are set to are listed below. The entire command is enclosed in quotes to prevent REXX from interpreting the parameter string and performing substitutions. Any delimiter character may be substituted in place of the slash (/), but the slash is recommended to make programs consistent. The command name EXTRACT and the variable names may be in upper or lower case (but upper case is the convention).

The EXTRACT command sets a non-zero return code (rc) if there is a syntax error in the command or if any of the variable names are invalid. An exception is that variable names of the form $xxx (where xxx is a SETV variable - see below) do not cause a nonzero return code, even if xxx is undefined.

## EXTRACT variables

Variables set by the extract command are governed by the following convention. For example, if the variable name used on the EXTRACT command is VAR, then the REXX variable VAR.0 will be set to the number of values associated with VAR. The variables VAR.1, VAR.2, ..., VAR.n (where n is the number returned in VAR.0) will be set to those values. The value returned in each VAR.i is a whole number or a character string.

For example, the command "EXTRACT /EOF/" in an editor macro sets REXX variable eof.0 to the number 1 and REXX variable eof.1 to the character string "ON" or "OFF", depending on whether or not the editor is positioned at End-of-File.

Below is a description of variables that can be used with the EXTRACT command. The minimum abbreviation for each variable name is shown in capital letters.

ACTion — Indicates whether changes have been made to the file and its attributes. Each variable is set to the character string ON or OFF.

- ACTION.0 — 4
- ACTION.1 — ON|OFF
  Indicates whether or not the text of the file has been changed since the beginning of the edit (or the last SAVE command). Value ON indicates there are unsaved changes.
- ACTION.2 — ON|OFF
  Indicates whether the file name has been changed by the NAME command since the beginning of the edit.
- ACTION.3 — ON|OFF
  Indicates whether the file tag has been changed by the TAG command since the beginning of the edit.
- ACTION.4 — ON|OFF
  Value ON is returned if the editor work file was too small to hold the edited file at the start of the edit. ON means that records will be lost if a SAVE or FILE command is done.

AUTOSKip — Returns the current settings for the AUTOSKIP command.

- AUTOSKIP.0 — 2
- AUTOSKIP.1 — ON|OFF
  Indicates whether AUTOSKIP INP is in effect.
- AUTOSKIP.2 — ON|OFF
  Indicates whether AUTOSKIP CMD is in effect.

CASE — returns the current case setting.

- CASE.0 — 2
- CASE.1 — UPPER|MIXED
- CASE.2 — RESPECT|IGNORE

CD — returns the current directory name. Returns "\" for the root directory, or else "\dirname" or "\dirname1\dirname2" etc.

| | |
|---|---|
| CD.0 | 1 |
| CD.1 | current directory (without trailing blanks). |

COLOR — returns the current color settings. The alternate spelling COLOUR may be used, in which case the returned variables are also COLOUR.i.

| | |
|---|---|
| COLOR.0 | 1 |
| COLOR.1 | BASE\|EXTENDED |

CSRPOS — returns the cursor position and screen size.

| | |
|---|---|
| CSRPOS.0 | 6 |
| CSRPOS.1 | The cursor column number when the 3270 action key was pressed, relative to the start of the record (not the display window). Value 0 is returned if the cursor was not in a line of the file (e.g. in the command area). Example: if the WINDOW is 11 to 50 and the cursor is at the 3rd character of the displayed text, then CSRPOS.1 is 13. |
| CSRPOS.2 | Cursor row number (absolute position on the screen). |
| CSRPOS.3 | Cursor column number (absolute position on the screen). |
| CSRPOS.4 | Screen size in rows. |
| CSRPOS.5 | Screen size in columns. |
| CSRPOS.6 | Number of screen rows used by the editor, not counting the SHOW text lines. For example, the command area is on row (CSRPOS.6)-1 of the screen. |

CURLine — returns the contents of the current line and information about its location in the file.

| | |
|---|---|
| CURLINE.0 | 3 |
| CURLINE.1 | line number in file. This is always 1 or more. If at End-of-File, (total number of lines) + 1 is returned. |
| CURLINE.2 | line number on screen |
| CURLINE.3 | contents of current line (null string if at End-of-File) |

DEFENTer — Gives the command string defined for the Enter key by the command DEFINE ENTER xxx, including trailing blanks. If no string is defined for Enter, a null (0 length) string is returned.

| | |
|---|---|
| DEFENTER.0 | 1 |
| DEFENTER.1 | The command string or null. |

DEFINPut — Gives the command string defined for Input Mode by the command DEFINE INPUT xxx, including trailing blanks. If no string is defined for Input Mode, a null (0 length) string is returned.

| | |
|---|---|
| DEFINPUT.0 | 1 |
| DEFINPUT.1 | The command string or null. |

DELCHAR — Returns the line delete character (normally ":"), or a blank if there is none. The delete character is defined by the DELCHAR command and is used by the MINSERT and MDELETE commands.

| | |
|---|---|
| DELCHAR.0 | 1 |
| DELCHAR.1 | "x" (where x is the delete character or a blank) |

DELIM — returns the command delimiter character (normally ";"), or a blank if there is no command delimiter.

| | |
|---|---|
| DELIM.0 | 1 |
| DELIM.1 | "x" (where x is the delimiter character or blank) |

EOF — returns ON or OFF depending on whether the current line pointer is at the end of file position.

| | | |
|---|---|---|
| | EOF.0 | 1 |
| | EOF.1 | ON\|OFF |
| FLAG | returns ON or OFF depending on whether modification flagging is active. | |
| | FLAG.0 | 1 |
| | FLAG.1 | ON\|OFF |
| FLScreen | returns the first and last line numbers on the screen. | |
| | FLSCREEN.0 | 2 |
| | FLSCREEN.1 | the line number (in the file) of first line displayed on the screen. |
| | FLSCREEN.2 | the line number (in the file) of last line displayed on the screen. |
| FName | returns the current file name, or a null string if no name is defined. | |
| | FNAME.0 | 1 |
| | FNAME.1 | current file name (without trailing blanks). |
| FSMODE | returns ON or OFF depending on whether 3270 full-screen (FS) mode is in effect. | |
| | FSMODE.0 | 1 |
| | FSMODE.1 | ON\|OFF |
| HEX | Return the settings for hexadecimal input (XIN/AIN commands) and output (HEX/ALPHA/BOTH commands). | |
| | HEX.0 | 2 |
| | HEX.1 | ON\|OFF<br>Indicates whether hex input is enabled. |
| | HEX.2 | HEX\|ALPHA\|BOTH<br>Indicates the setting for hex output. |
| KEYHIT | Indicates which 3270 action key was pressed. This applies to full-screen (FS) mode only). | |
| | KEYHIT.0 | 1 |
| | KEYHIT.1 | The PF key number, or 0 if the Enter key was pressed. |
| LENgth | Returns the length of the ZONE part of the current line excluding trailing blanks, or 0 if at Top-of-File or End-of-File. The "zone" is columns 1 thru n, where n is set by the command "ZONE n". In full-screen mode on an 80 character wide screen, the zone is columns 1 thru 72, by default. | |
| | LENGTH.0 | 1 |
| | LENGTH.1 | Length of ZONE part of current line excluding trailing blanks. |
| LIne | Returns the line number of the current line in the file. This is always 1 or more. If at End-of-File, (total number of lines in the file) + 1 is returned. | |
| | LINE.0 | 1 |
| | LINE.1 | current line number |
| LOCSTR | returns the last locate string that was issued with a LOCATE, FIND, SEARCH, HUNT, ULOCATE, or UFIND command. | |
| | LOCSTR.0 | 1 |
| | LOCSTR.1 | last locate string, or null if no previous locate operation was done. |
| LOG | Returns information about editor logging. Logging is done to enable an edit restart if failure occurs. | |
| | LOG.0 | 2 |
| | LOG.1 | ON\|OFF<br>Indicates whether logging is enabled. |
| | LOG.2 | The log file id number. This is a number from 1 to 9, if logging is on. It identifies which log file is being used. |

LRecl — Returns the logical record length in effect for the edit session.

| | |
|---|---|
| LRECL.0 | 1 |
| LRECL.1 | Record length. |

MArk — Returns the starting and ending line numbers of the marked area. If no lines are marked, zero is returned.

| | |
|---|---|
| MARK.0 | 2 |
| MARK.1 | starting line number, or 0 if no marked lines. |
| MARK.2 | ending line number, or 0 if no marked lines. |

MSG — Returns the text of the last message line put out by the editor. Note that if the complete message was put out as 2 or more lines, only the last line is returned. The text is returned even if the message display was suppressed by the MSGS OFF command.

| | |
|---|---|
| MSG.0 | 1 |
| MSG.1 | text of last message. |

MSGS — Returns OFF if editor messages are currently suppressed by the MSGS OFF command, otherwise returns ON.

| | |
|---|---|
| MSGS.0 | 1 |
| MSGS.1 | ON\|OFF |

NEWFILE — Returns ON if the NEW option was used on the EDIT command (or the file name was NULFIL). This means that the editor started without any data.

| | |
|---|---|
| NEWFILE.0 | 1 |
| NEWFILE.1 | ON\|OFF |

NULLs — returns ON if 3270 screen fields have trailing nulls (rather than blanks).

| | |
|---|---|
| NULLS.0 | 1 |
| NULLS.1 | ON\|OFF |

NUMber — returns ON if line numbering is active, that is, line numbers are displayed.

| | |
|---|---|
| NUMBER.0 | 1 |
| NUMBER.1 | ON\|OFF |

PFn — Returns the current program function key definition for function key *n*. Where *n* is a number between 1 and 24. If "*" is specified for *n* instead of a number, (e.g. "EXTRACT /PF*/"), the definitions of all 24 function keys are returned in the variables PF1.i, PF2.i, ..., PF24.i.

| | |
|---|---|
| PFn.0 | 2 |
| PFn.1 | DEFINED\|UNDEF Indicates whether the function key is defined. The returned string is 8 characters long, with trailing blanks. |
| PFn.2 | Function key definition, including trailing blanks. If the key is not defined, a null string is returned. |

PREFIX — Returns whether the PREFIX command is active, and other information relating to prefix operations.

| | |
|---|---|
| PREFIX.0 | 3 |
| PREFIX.1 | ON\|OFF indicating whether PREFIX command is active. |
| PREFIX.2 | The color number for the prefix area, as set by the COLOR command. |
| PREFIX.3 | Is "ON" if a prefix operation was executed during this same screen interaction, or if a macro xxx (for "DEFINE INPUT xxx") was invoked to process Input mode lines during this interaction. "ON" indicates that the new current line is as set by the prefix operation or Input mode macro, rather than by the cursor position when the action key was pressed. |

RDONLY — Returns ON if the editor is in read-only mode (i.e. the BROWSE command, or command subset 2).

RDONLY.0 1
RDONLY.1 ON|OFF

RECFM Returns the record format for the edit. This is the record format of the file being edited, or the record format specified on the EDIT command. It is the record format that will be used when a file is written by the FILE or SAVE command. If a UDS (user data set) is being edited, or the UIO option was used on the EDIT command, the returned record format is always FC.

RECFM.0 1
RECFM.1 F|FC|V|VC

SHOW returns information about SHOW text. This is the text displayed by the SHOW command at the bottom of the screen.

SHOW.0 6
SHOW.1 actual number of SHOW lines, or 0 if NOSHOW.
SHOW.2 file name used on SHOW command, or null string if none.
SHOW.3 starting line number used on SHOW command, or 1 if none.
SHOW.4 ending line number used on SHOW command, or 99999999 if none. 99999999) if none
SHOW.5 color number for non-highlighted show text, as set by the COLOR command.
SHOW.6 color number for highlighted show text, as set by the COLOR command.

SIZe returns the total size of the file in terms of the number of records. This is the size of the editor's current working copy of the file, not the original file.

SIZE.0 1
SIZE.1 number of records.

SUBSET Returns the command subset number, as set by the SUBSET command. A nonzero subset number may indicate that some commands or command operands are not allowed.

SUBSET.0 1
SUBSET.1 Subset number.

TAG Returns the current file tag, from the original file or from the last TAG command. A null string is returned if there is no tag or the tag is unknown.

TAG.0 1
TAG.1 File tag (64 characters) or a null string.

TERMinal returns DISPLAY or TYPEWRITER depending on the terminal used.

TERMINAL.0 2
TERMINAL.1 DISPLAY|TYPEWRITER
DISPLAY indicates a 3270-type terminal (able to accept 3270 data streams and able to use editor full-screen mode). TYPEWRITER indicates some other type of connection.
TERMINAL.2 BATCH|TERM
BATCH indicates the editor is running as a batch job.

TOF returns ON or OFF depending on whether the current line pointer is at the top of file position.

TOF.0 1
TOF.1 ON|OFF

UDSEDIT returns ON if a UDS (user data set) file is being edited.

UDSEDIT.0 1
UDSEDIT.1 ON|OFF

| | |
|---|---|
| UIO | Returns ON if the UIO option was used on the EDIT command.<br>UIO.0 1<br>UIO.1 ON\|OFF |
| WASINPut | Returns ON if the editor was in full-screen Input Mode when the 3270 action key (Enter or a function key) was pressed.<br>WASINPUT.0 1<br>WASINPUT.1 ON\|OFF |
| Window | Returns the first and last column numbers of the text display area. These column numbers are relative to the file record, not to the screen.<br>WINDOW.0 2<br>WINDOW.1 Starting column number in record.<br>WINDOW.2 Ending column number in record. |
| Zone | returns the number of columns of the current zone setting, as set by the ZONE command.<br>ZONE.0 1<br>ZONE.1 current zone value. |
| $xxx | An EXTRACT name starting with $ returns the string assigned to the corresponding SETV name. xxx is the 1 to 8-character name used on the editor command "SETV xxx yyy". The EXTRACT command sets the Rexx variable xxx to yyy. Note that no ".n" numeric suffix is used. This allows editor macros to remember things between calls (but only within the same edit session). For example, if "setv abcd this value", then "EXTRACT /$ABCD/" sets Rexx variable ABCD to "this value". If xxx is not currently defined by a SETV command, EXTRACT sets xxx to a null (zero length) string. |

**Example**

The following sets the variables FNAME, CASE, and HEX.

```
'EXTRACT /FNAME/CASE/HEX/'
```

# Sample Macros

The following sample macros are available for inspection in the files name.MAC, where "name" is the macro name. The files may be slightly different from the listings shown here. Other interesting sample macros are CALC, CURFIRST, FILEINFO, XL.

1. GET macro.

   The GET macro saves the current file being edited and brings in a new file to be edited.

```
parse arg fname
"EXTRACT /ACTION/"
if action.1 = "ON" then do
   say ' File has unsaved changes. Do you wish to save them?'
   pull resp
   if substr(resp,1,1) = "Y" then "SAVE"
end
"NAME" fname
"TOP"
"DEL 9999999"
"MERGE" fname
rcsave=rc
"TOP"
exit rcsave        /* exit with return code from MERGE */
```

2. BROW, BU, and BD Macros.

   The following three macros (BROW, BU, and BD) are used to allow browsing a file using the show command, while editing another file. Note the use of PF key definitions to preserve information between calls to the macros. (The SETV command could also be used.) The NOSHOW command can be used to cancel this.

   BROW issues the initial show command and sets up PF1 and PF2 to scroll up (BU) and down (BD).

```
/* BROW - init edit browse */
PARSE ARG FNAME
DEF PF1 BU FNAME 1
DEF PF2 BD FNAME 1
SHOW FNAME 1 10
```

   BU scrolls up and redefines the PF keys for next time.

```
/* BU - do edit browse up */
parse arg fname n1
n1=n1-10
if n1<1 then n1 = 1
n2 = n1+9
def pf1 bu fname n1
def pf2 bd fname n1
show fname n1 n2
```

   BD scrolls down and redefines the PF keys for next time.

```
/* BD - do edit browse down */
parse arg fname n1
```

```
n1=n1+10
n2 = n1+9
def pf1 bu fname n1
def pf2 bd fname n1
show fname n1 n2
```

3. The KEYS macro provides a full screen approach to determining and modifying the current function key settings in the EDITOR. The panel used by keys is in the file PFSCR.

```
/* This sample macro allows the user to inspect and modify the
   program function key definitions. The new definitions are
   saved in the user's EDITOR file if required */
'EXTRACT /PF*/'
j=0
do i=1 to 24 by 2
  j=j+1
  pfx='pfscr.' || i || '=pf' || j ||'.2'
  pfy='pfscr.' || i+1 || '=pf' || j+12 ||'.2'
  interpret pfx
  interpret pfy
end
do forever
  'PANEL PFSCR'
  if aid='A1' then exit
  if aid=3 then do
    call getpfk
    call setpfk
    exit
  end
  if aid=10 then do
    call getpfk
    call setpfk
    call savpfk
    exit
  end
end
getpfk:
  j=0
  do i=1 to 24 by 2
    j=j+1
    k=j+12
    l=i+1
    pfk.j=pfscr.i
    pfk.k=pfscr.l
  end
return
setpfk:
  do i=1 to 24
    'DEF PF'||i pfk.i
  end
return
savpfk:
  'MUSIO READ *USR:EDITOR ALL'
  'MUSIO CLOSE *USR:EDITOR'
  if queued()=0 then do
    queue '/SYS REG=256'
```

```
    queue '/FILE MACLIB PDS(*.MAC)'
    queue '/INCLUDE *COM:EDITOR'
    queue 'REX ON'
  end
  j=queued()
  k=0
  do i=1 to j
    pull x
    if substr(x,1,3)¬='DEF' then do
      k=k+1
      rec.k=x
    end
  end
  do i=1 to k
    queue rec.i
  end
  do i=1 to 24
    queue 'DEF PF'||i pfk.i
  end
  'MUSIO WRITE *USR:EDITOR ALL'
  'MUSIO CLOSE *USR:EDITOR'
return
```

# Defining Prefix Macros

The prefix area of the Editor is turned on with the PREFIX ON command. There are several commands already defined for this area. Details about using the prefix area can be found earlier in this chapter under the heading "Prefix Area".

The standard prefix commands (I, D, C, MM, etc.) are predefined. You would use the DEFINE PREFIX command only to add your own prefix operations, or to undefine or rename existing ones. When parameters tt and mmmmmmmm are omitted, the prefix command pppp is made undefined.

The types tt for DEFINE PREFIX are:

| | |
|---|---|
| 1 | Non-paired command, with no target required (e.g. In, Dn) |
| 1T | Non-paired, requiring a target (e.g. Cn, Mn) |
| 1L | Same as 1, except the operation is done last, after other prefix operations (e.g. /). |
| 2 | Paired, no target (e.g. DD, >>) |
| 2T | Paired, requiring a target (e.g. CC, MM) |
| 2L | Same as 2, except the operation is done last, after other prefix operations. |
| F | Target, of type "following" (e.g. F) |
| P | Target, of type "preceding" (e.g. P) |

You can use REXX to define your own prefix commands. For example, suppose you wanted to use XX to print a group of lines. XX is entered in the prefix area of the first and last lines of the group. You would define it by:

```
define prefix xx 2 myxxmac
```

"myxxmac" is the name of the macro to be invoked (any 1 to 8-character macro name), and "2" is the type of prefix operation (in this case, a "paired" command). You would have to write the editor macro in Rexx, and store it as file MYXXMAC.MAC. The macro would do whatever is needed to print the lines. It would be

invoked by the editor as:

```
MYXXMAC n1 n2 0 0 XX op
```

n1 and n2 are the line numbers of the first and last lines of the group; XX is the prefix command name; op is the parameter (if any).

The prefix command name can contain special characters. For example, the . and .xxx prefix commands are defined by:

```
define prefix . 1 $$point
```

When the prefix command starts with a special character (a non-letter), the command name is ended by the first letter, digit, or blank encountered in the prefix area. For example, /10 and /e both have "/" as the command name, and the rest ("10" or "e") as the parameter.

If the prefix command name on the DEFINE command is already defined, the new definition replaces the old one.

To make a prefix command name undefined, omit the type and macro name parameters. Example: define prefix ff

The pre-defined prefix operations use macro names $$xx, where xx is the command name, e.g. $$I.MAC, $$MM.MAC, ...

The Rexx macro for a prefix operation is invoked with a standard set of parameters, which completely define the parameters for the operation. When the operation (such as CC or MM) involves more than one entry, the macro is invoked only once. The macro is invoked by the following command line:

```
macname num1 num2 targetnum targetcmd cmdname parameter
```

Where:

| | |
|---|---|
| macname | is the name of the macro. |
| num1 | is the line number (8 digits) of the first (or only) line of the file on which the operation acts. |
| num2 | is the line number (8 digits) of the last line of the file on which the operation acts, or is 0 when only one line is involved. |
| targetnum | is the target line number, or 0 if no target is involved. The target line number has already been converted to F ("following") form, by subtracting 1 if a P-type ("preceding") prefix command was used. However, when P is entered on the Top-of-file line, 0 is used (not -1). |
| targetcmd | is the prefix command name (normally P or F) by which the target was specified, or 0 if no target is involved. The name is in upper case. |
| cmdname | is the prefix command name entered, in upper case. |
| parameter | is the parameter of the prefix command, as entered (if any). For example, for In or >>n, the parameter is n. |

Line number 0 refers to the Top-of-file line. A line number one more than the number of lines in the file refers to the End-of-file line.

For examples of prefix macros, see files $$I.MAC, $$C.MAC, and $$CC.MAC.

# Editor Restart Facility

## Introduction

A traditional problem with interactive systems has been the effect of system and terminal failures on users. Often much work and input data is lost when the system goes down or the user's terminal is disconnected.

The restart facility of the Editor addresses this problem. The idea is to record all edit activity (commands, input lines, etc.) in a log file, so that if the edit session is terminated abnormally, the changes can be reapplied to the original file when the user signs back on the system.

The restart feature is active by default, and is usually transparent to the user (until a restart is needed).

## General Description

At the start of each edit session, the Editor opens a log file (a new file is created if one does not exist). This file is stored in the user's library, and is named @ELOG.sss or @ELOG.sss.n, where *sss* is the user's subcode and *n* is a number from 2 to 9. Some special characters such as the / in the subcode are changed to 0. During the edit session, all Editor commands, input lines, 3270 full-screen changes, etc. are written sequentially to the log file. At normal termination of the edit session, the data in the log file is cleared (although the log file itself is not deleted). A log file name of the form @ELOG.sss.n is used if the normal file (@ELOG.sss) is in use by another edit session on the same userid and subcode.

If the edit terminates abnormally, say because of a system failure or a terminal disconnect, the log file is not cleared. The next time the user starts an edit, using the same userid and subcode as before, the Editor notes that log file data already exists (i.e. the log is not empty) and tells the user that the previous edit may be restarted. The user has the choice of stopping the new edit and restarting the previous one (YES or Y response), or of clearing the log file and continuing the new edit with a fresh log (NO or N response). Note that after a NO response, the log data is gone and thus the restart cannot be done later.

If the user chooses to restart the previous edit, the Editor executes a restart program. This utility program examines the log file and generates an Editor job (in save file @EXEC) which will redo the old edit session up to the point of the failure. The starting point for this Editor job is the last successful FILE command in the edit session, or else the beginning of the session if no successful FILE was done.

The generated Editor job is then executed. This reapplies the Editor commands and file changes, using a special mode of the Editor called *restart mode*. Once the restart is complete, the user is in regular command mode, at the approximate point of the original failure, and may continue editing. Subsequent commands and input lines are added to the old log data. Also, the @EXEC file containing the generated Editor job is deleted.

Note that if the Editor is cancelled during MUSIC attention mode (ATTN), using the /CANCEL command, this is equivalent to an abnormal end, and the next edit will ask whether or not a restart should be done. If the /CANCEL was intentional, simply reply NO to the question.

Logging is not done if the BROWSE command is used, or if any of the option RO, UIO, or NOLOG is used on the EDIT or TEDIT command.

# Extent of Recovery

Recovery of the edit session when an edit restart is done should be complete except for the last few commands and input lines entered before the failure. These last lines are not written to the log file because they are still in a main storage buffer at the time of the failure, even though they have been processed by the Editor. The maximum number of log lines lost in this way may be set by the LOG command, described in the next section. The default value is 25 lines for 3270-type terminals and 20 lines for other terminals.

Also, in Editor Input mode (in other than 3270 full-screen mode), the Editor gets control only after each *n* lines of input (where *n* is the number set by the LOG command, or 20 by default). This is because the Editor does spooled conversational reads when in input mode (refer to the section "Spooled Conversational Reads" in *Chapter 4. File System and I/O Interface*. For this reason, up to n additional lines may be lost if failure occurs during Input mode.

On a 3270-type terminal in full-screen mode, screen changes are transmitted to the system only when an action key (such as ENTER or a program function key) is pressed. Therefore all changes entered on the screen since the last action key are always lost.

# The LOG Command

The format of the LOG command is:

```
      LOG
or    LOG n
or    LOG END
or    LOG PURGE
```

When used without a parameter, the LOG command forces any log lines being held in main storage buffers to be written to the log file. This ensures that all edit activity prior to the LOG command will be available for restart if necessary.

When used with a number *n* as a parameter, the command defines the maximum number of log lines which will be held in main storage. Initially, the Editor assumes a value of 25 lines, i.e. "LOG 25", for 3270-type terminals and 20 lines for other terminals.

The number n specified on the LOG command is also the maximum number of lines which may be entered in Input mode (in non-full-screen mode) before the Editor gets control. This may be noticeable as a slight pause after each group of n lines entered in Input mode.

Using LOG without a parameter does not change the value n.

As explained in the previous section, additional lines may be lost if the Editor is in Input mode or Full-Screen mode.

If the parameter END is specified, the Editor will clear and close the log file. The log file can then be purged by issuing the command "PURGE @ELOG.sss" to save file space. (*sss* is the user's subcode. Special characters such as the / in the subcode should be replaced by 0).

If the parameter PURGE is specified, it is equivalent to specifying the parameter END and the PURGE command as mentioned above. That is, the log file will be cleared, closed, and purged all at the same time.

## Effect on Performance

Apart from the restart process itself, the overhead due to the Editor restart feature is the extra input/output involved in opening and writing to the log file during an edit session. The amount of input/output to the log file is largely controlled by the value *n* used in the "LOG n" command. The smaller the value of *n*, the more input/output and the more overhead. Thus there is a trade-off between efficiency and the amount of data lost when a failure occurs.

With a value for *n* such as 20, the extra overhead should not be a problem, since log file input/output will normally occur only once every few Editor interactions, and each interaction is a time slice involving considerable input/output anyway.

## Suppressing the Restart Feature

If you wish the Editor not to create a log file or look for an existing one, use the NOLOG option on the EDIT or TEDIT command. This may be useful if you need to edit something without disturbing the existing log file. For example:

```
EDIT FILEXYZ NOLOG

TEDIT TEXTFILE NEW NOLOG
```

To suppress logging and restart for an edit of a user data set (UDS) file, use the following control statement at the beginning of the job:

```
/PARM *UDS NOLOG
```

When the Editor is run from batch, logging and restart are never done.

## Restrictions

- In order to restart an edit, the user must sign on with the same userid and subcode as before.
- If the user has reached the file space limit for the userid, the Editor will not be able to create a log file, and logging will not be done. Logging is also terminated if the log file becomes full and space cannot be added to it. Similarly, a restart may fail if the restart utility program cannot create the file @EXEC because the save file space limit has been reached.
- If a file is specified on a MERGE command, and the file is later purged (deleted) or renamed during the same edit session, the restart may not be successful. One way to avoid this problem is to do a SAVE command after the PURGE or RENAME command.
- If an edit job is run using a /FILE statement for unit 3, and a command such as "MERGE (3)" is used to merge lines from unit 3 into the file being edit, then the edit session cannot be correctly restarted. This is because the generated Editor job will not have a /FILE statement for unit 3, and the MERGE command will fail during restart mode.
- If column number limits are used on an Editor FILE command, the command may still be used as a restart point. This may cause the unsaved columns to be lost when a restart is done.

# Sample Terminal Session

In the following example, the user begins by editing the file MYFILE. During the edit, the system goes down before any changes are saved. When the system comes back up, the user signs on again and tries to use the Editor. The Editor sees that log data already exists, and allows the user to restart the original edit. The user then continues editing MYFILE.

Lines entered by the user are shown in lower cases, while messages from the system are in upper case.

```
/id ccccsss    (the user signs on)
...
edit myfile
...
...  (edit commands, etc.)
...
---------- (the system goes down at this point) ----------
...
/id ccccsss    (the user signs on again, same code and subcode)
...
edit myfile    (or edit of any other file)

*** LOG FILE EXISTS FROM PREVIOUS EDIT, AND MAY BE
    USED FOR RESTART (RECOVERY OF UNSAVED DATA).

DO YOU WISH TO CONTINUE THE PREVIOUS EDIT?  (ANSWER YES OR NO)
?
yes

-- EDIT RESTART --     CODE,SUBCODE = CCCC SSS

EDIT-LOG FILE REPRESENTS EDIT AT 14:13 (TODAY)
OF FILE MYFILE

ENTER A BLANK LINE TO PROCEED (OR ENTER /CANCEL TO STOP)
?
               (user enters a blank line)

EDIT SESSION WILL BE RESTARTED FROM THE BEGINNING.

AN EDIT RESTART JOB HAS BEEN GENERATED AND
WILL NOW BE EXECUTED.  ONCE THE RESTART HAS BEEN
DONE, YOU WILL BE IN COMMAND MODE.  PLEASE CHECK THE
FILE NAME AND CONTENTS BEFORE SAVING IT.


...
...  (messages, etc. from re-executed Editor commands)
...

*** RESTART COMPLETED.  PLEASE CHECK YOUR FILE.
...
...  (the user continues the interrupted edit of MYFILE)
...
```

# Editor Restart for Multiple Sessions

If multiple sessions are started on the same terminal, or if the same user code and subcode is signed on to more than one terminal, the Editor uses an alternate log file name if the normal log file is in use. The normal (primary) log file name is @ELOG.sss, where *sss* is the subcode. Alternate log file names are @ELOG.sss.n, where *n* is 2 to 9. @ELOG.sss.2 is used if @ELOG.sss is busy. @ELOG.sss.3 is used if @ELOG.sss.2 is busy, etc.

If more than one log file contains restart data when an edit restart is done, the user is asked which edit session to restart. Note: the Editor looks for restart data only in the primary log file. If the primary log file is empty or does not exist, the user must type RESTART.EDIT when in *Go mode in order to do secondary restarts.

MUSIC/SP User's Reference Guide

Part 4 - Chapter 8 (UR_P4.PS)

# Chapter 8. Processors

# Overview of Processors

To run a program written in a high-level language, the system goes through several steps before the program actually starts running. The high level language input is referred to as the "source" language. The desired result of a compilation is machine language code, or *object code*, which is placed in a file known as the *object module*.

The first step is the *compile* step. Here a compiler checks over your *source* statements and the compiler may give you a *source listing* of them. You use a compiler that was specifically designed to handle the particular computer language that you are using. Any errors that the compiler finds are identified (or *flagged*) so that you can correct them. These error messages may appear within your source listing or they may appear later under a heading of *diagnostics*. The compiler produces what is known as *object code* based on your source statements and this object code is kept for use in the next step. You can get a copy of this object code if you want in a format known as an *object module*. (The object modules cannot be converted back to source programs. Thus you should not discard your source just because you now have an object module.)

The next step is the *load* step. Here the loader receives the object code produced by the compiler and loads it into main storage. Any system-supplied subroutines that your program needs are also loaded into main storage at this time.

The last step is the *execute* (GO) step. This is where the program is given control. The program communicates with the system through a set of system routines called the *user-system interface*.

*Note:* FSI (Full Screen Interface) provides a menu option for creating programs and using processors.

## MUSIC Compilers

The MUSIC compilers automatically start the chain of events through the steps described above without you having to be aware of them. If the compiler or loader notices some errors in your program, then it stops the chain so that you can correct the errors and start again.

The name on the /LOAD statement specifies the required processor. This may be the name of a compiler, loader, interpreter, or other processor.

APL, VS APL, BASIC, and REXX operate differently from compilers and are considered *interpreters*. GDDM is a collection of subroutines and utilities used in conjunction with a compiler.

Figure 8.1 below illustrates the steps involved in processing a program through a compiler.

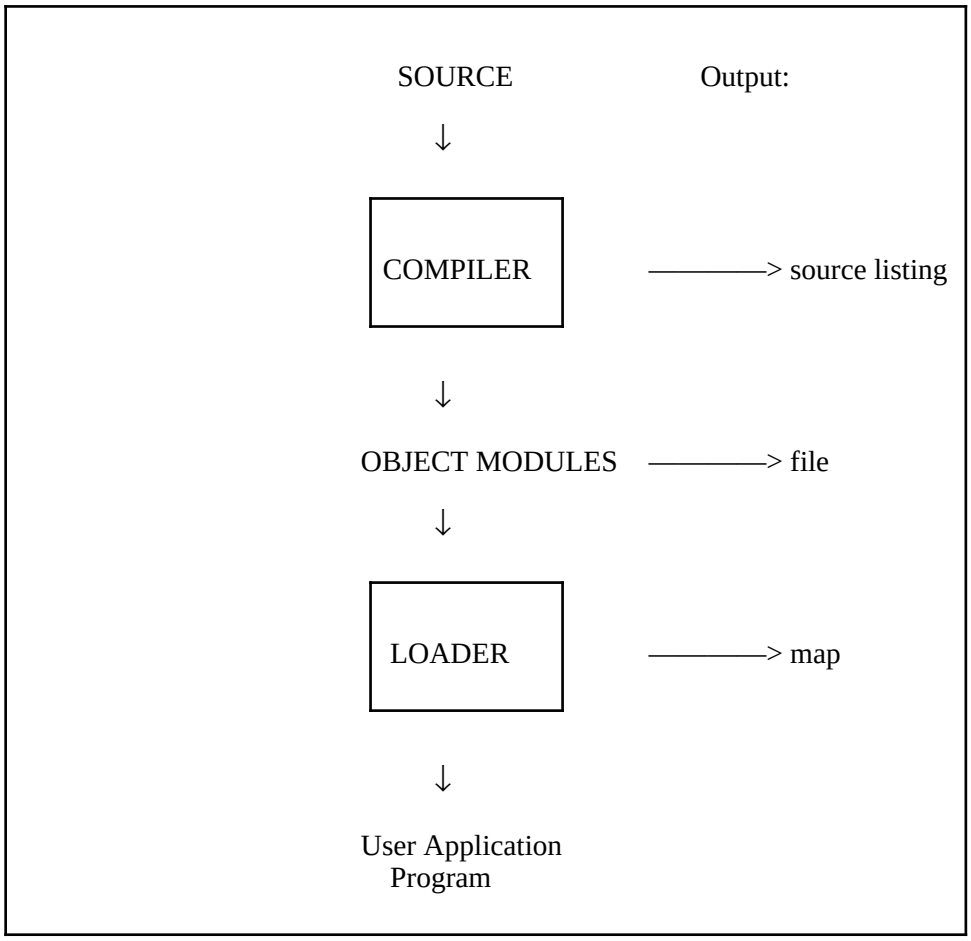



*Figure 8.1 - Compilers*

# MUSIC Loaders

If you have object modules produced by a compiler, then you can bypass the compiler step and proceed directly to the loader step. You normally accomplish this by specifying a /LOAD ASMLG statement except in the case of PL/I. PL/I requires its own loader, /LOAD PLILG.

If parts of your program are in source form and other parts in object modules, then just use the compiler. MUSIC compilers will separate out the object modules and cause them to be loaded at the appropriate time.

# MUSIC Linkage Editor

Instead of loading the object modules directly into main storage, you have the option of creating a *load module* with the MUSIC Linkage Editor and stored in a file. The MUSIC Linkage Editor is called by a /LOAD LKED. There are two main reasons why you might want to use the Linkage Editor. One is that your program is too big for all of it to fit into the MUSIC user region at one time. In this case, you use an overlay structure to split your program into segments that will each fit into the user region. The other main use of the Linkage Editor is to achieve a faster loading time for programs that are used often. You will find that it is normally faster to load a load module than loading the program from the corresponding object modules.

Figure 8.2 shows the steps involved in creating a load module with the linkage editor.

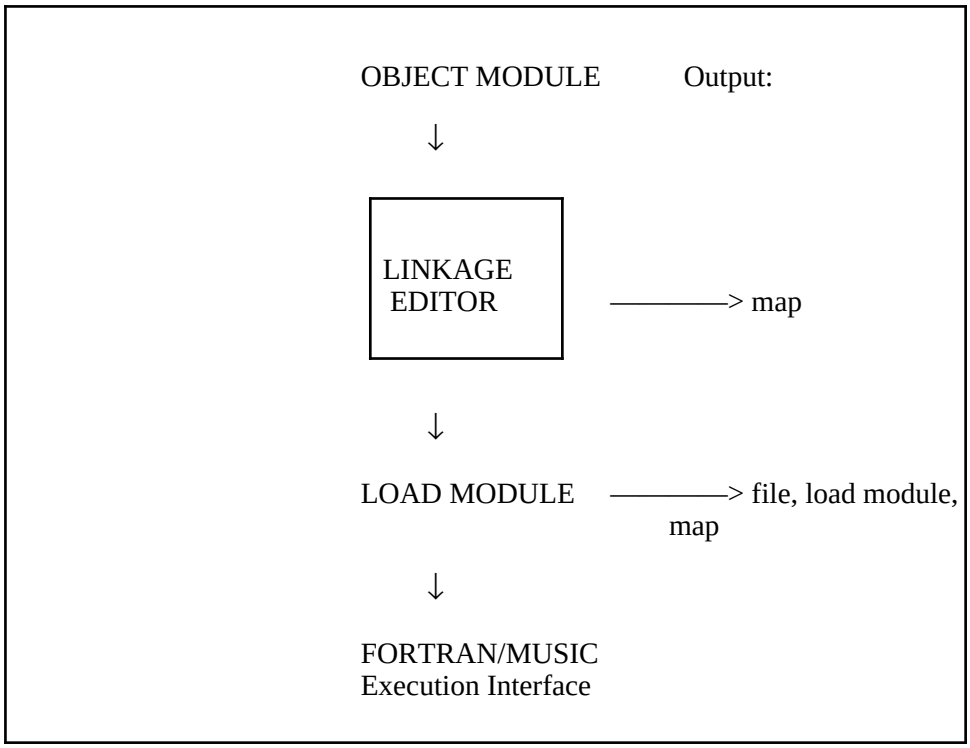



*Figure 8.2 - Linkage Editor*

# Load Module Executors

Load module executors are used to execute a program that has been previously created with the MUSIC Linkage Editor. The loading process can be considerably faster than the use of a standard loader. The names of the two load module executors on MUSIC are: XMON and XMPLI. For a load module produced by an object module from a PL/I program, use the /LOAD XMPLI statement. You must use the /LOAD XMON statement if the load module is produced by an object module from a COBOL or VS Assembler program. The OS/MUSIC interfaces will be loaded in the last two cases.

Object modules are used to form the load modules. However, once you have a load module you cannot transform it back into object modules.

# MUSIC Interpreters

An interpreter takes as input a source language. However, unlike a compiler, an interpreter does not generate machine language code in an object module. Instead, the source language statement is executed immediately. One way of thinking about it is that the machine language instructions for the current source language statement are generated, then executed, and then discarded.

Processors, such as REXX, APL, and BASIC, are interpreters, and as such do not adhere to the sequence of events described above in "MUSIC Compilers". Refer to the individual descriptions for complete information on how to run these processors on MUSIC.

# Programming Tutorials

MUSIC provides a tutorial facility for learning programming languages. To invoke this facility type "TUT" in *Go mode or choose the "Tutorials" item on the FSI main menu. Help is provided once this facility is invoked.

# APL - Subset Version

MUSIC supports two versions of APL. The subset APL version described below should be used only if your installation does not have VS APL (Program Product 5748-AP1).

The MUSIC/APL Subsystem is an implementation of the IBM APL S/360 Type III program. APL is a conversational time-sharing facility based on a mathematical programming language called "A Programming Language". MUSIC/APL runs under MUSIC in much the same way as any other program. The entire MUSIC/APL session is regarded by MUSIC as one job.

The APL language is concise and has a small number of syntax rules. It supports built-in mathematical operators which permit vector and matrix manipulation. It operates as both a sophisticated and powerful desk calculator and as a programming language.

## Running MUSIC/APL

The user invokes MUSIC/APL by typing APL when the workstation is in command (*Go) mode. (Some installations may require you to type /EXEC APL if your user profile is set up not to allowed the implied EXEC feature.) To end the APL session, the user types the APL command )OFF. This APL command will return the user to MUSIC command mode. The APL session can also be terminated by the user typing in the MUSIC command /CANCEL.

A MUSIC execution time limit applies for your entire APL session. MUSIC/APL has a *limiting time* feature which allows user notification when excessive function evaluation time is detected. This feature is discussed in more detail later in this writeup, together with a full description of how the APL session time limit is set.

## Workstation Requirement - APL

MUSIC/APL can be run from most workstations supported by MUSIC. However, it uses a different set of characters to those normally available. Some workstations have an optional APL character set feature. Operation of APL from workstations without the APL character set is not recommended. The following topics discuss the operation of APL on the different types of supported workstations.

### APL on 2741 and 3767 Terminals

These APL characters are available on an IBM 2741 terminal through the use of an interchangeable type element. This element is available for both the EBCD keyboard arrangement (#1167988) and the correspondence arrangement (#1167987). The last three digits of the part number is stamped on the top rim of the type element.

The APL character set for an IBM 3767 terminal is available through the alternate character set feature.

The ATTN key on these terminals is handled according to the usual APL rules. These rules are somewhat different to the usual MUSIC usage of this key.

## APL on 3270 Series Terminals

The APL character set is available on the 3270 series display terminals through the APL character set feature. To inform MUSIC that your 3270 terminal has this feature, you must use the terminal class specification of 3270A, if it is a 3277 terminal, or use 3270B, if it is a 3278 or 3279 terminal, when you sign on to MUSIC. See the /ID command for details on how to specify the trmcls parameter.

When installed, the special APL ON/OFF key is used to select the APL character set used for entry of characters. The terminal can always display APL characters. The PA1 key is used in the usual MUSIC manner to signal attention when the terminal is in WORKING mode. Escape from input in APL is through the OUT sequence signalled via the PA1 key when the terminal is in READING mode. The PA2 key is used in the usual MUSIC manner to signal that the next page is to be displayed.

## APL on TTY Terminals

Some TTY terminals have an APL characters available on them. MUSIC/APL may work correctly from these devices, though you must check this out with the one you want to use. Since various manufacturers have established different conventions in APL mode, the user must indicate which one is to be used. The TRT option described below is used to specify the scheme used. (The default value of 4 suits a large number of terminals.)

```
/OPT TRT=n    (n is 1,2,3 or 4)
```

To verify that you have chosen the right number for the TRT option, execute the function KEYBOARD contained in the MUSFNS workspace. You should get a printout like the one shown in the example at the end of the VS APL discussion of this chapter of the manual.

The TTY terminals should be set to transmit even parity.

Some TTY terminals can generate a few more characters than are defined for an IBM 2741 terminal. These additional characters may be rejected if an attempt is made to enter them in APL mode.

TTY terminals have no direct equivalent of the ATTN key on an IBM 2741 terminal and so the following should be done instead: Use the LINE FEED key instead of the ATTN key to correct typing errors. Use the BREAK key instead of the ATTN key in all other cases.

# Usage Notes - APL

APL workspace numbers follow a convention described under the heading "APL Workspace Concepts".

MUSIC/APL workspaces are transferable to other MUSIC/APL systems. APL workspaces generated on other systems such as OS, CMS cannot be installed on MUSIC/APL.

MUSIC/APL handles the ATTN key according to the usual APL rules. These rules are somewhat different to the usual MUSIC usage of this key.

It is advisable to periodically save your current APL workspace since its contents will be lost should your workstation become disconnected from the computer for any reason.

Each input line is checked for the MUSIC command /CAN (the abbreviation of the /CANCEL command), and if found will terminate the APL session.

# APL Workspace Concepts

Your APL programs or functions are run in an area known in APL as your *workspace* (abbreviated WS). The contents of your active workspace may be saved as part of your *workspace library*. Another workspace may be loaded from this library to become the new active workspace.

The APL workspace library on MUSIC is contained on one or more User Data Set (UDS) files. The APL workspace number is used to indicate which APL workspace library is to be used. If this number is not specified, then library number 1 is assumed. The library number corresponds to the MUSIC I/O unit number as defined on the MUSIC /FILE command. The workspace size is fixed at 36,000 bytes.

Library number 3 contains the APL workspaces that are provided with the MUSIC system. These workspaces contain functions and information of common interest to MUSIC/APL users. The contents of this workspace are described later on in this writeup.

# Initializing an APL WS

You can make extensive use of APL without ever having your own workspace. However, if you want to save your workspace you must set up a MUSIC UDS file to hold workspaces. The following procedure illustrates how you can initialize a MUSIC UDS file so it can be used as an APL workspace library. This procedure is done only once for each UDS file you wish to use. Each UDS file can hold many workspaces, though you will usually find that 1 or 2 is sufficient for most work. The NREC parameter on the /FILE statement should be the number of workspaces times 284. Thus NREC(568) is used to allocate space to hold 2 workspaces. A sample initializing job follows:

```
/FILE 1 UDS(dsname) NREC(568) VOL(vvvvvv) NEW
/INCLUDE APLINT
```

The *dsname* is the dsname of you file and the *vvvvvv* in this example is the disk volume that contains your workspace library. Refer to the description on the /FILE statement in *Chapter 6. MUSIC Job Control Statements* for more details about this control statement.

# Using A WS Library - APL

You use the following set up to invoke MUSIC/APL with your workspace library:

```
/FILE 1 UDS(dsname) VOL(vvvvvv) OLD
/LOAD APL
```

The /FILE statement uses the same dsname and volume name that you used in the job to initialize the file. Note that the NREC parameter is not given and that the OLD is used instead of NEW.

The above set of commands can be saved on the MUSIC Save Library to make it more convenient for you to use them again at a later time. Thus, if you saved it under the name MYAPL, then all you need do is type the name MYAPL to run it at a later time.

In the above example, the workspace library has been shown as being assigned as MUSIC unit number 1. You may also use MUSIC unit numbers 2 and 3 for more APL workspace libraries. Normally you do not use unit number 3 for one of your own libraries since this will take the place of the system supplied one. You use the workspace unit number in the various APL system commands to tell APL which workspace library you want it to refer to. If not specified, then unit number 1 will be used.

# APL Public Workspaces

A MUSIC/APL user can access a library of public workspaces to do various functions. This library of workspaces is accessed as library number 3. A list of all the workspace names can be obtained by the APL command )LIB 3. To find out more about a specific one simply load it into the active workspace by the APL command )LOAD 3 xxx and then type DESCRIBE.

The following is a summary of the contents of the MUSIC/APL library:

| | |
|---|---|
| NEWS | This workspace contains information specific to APL under MUSIC such as limitations, hints and summary of the contents of the other workspaces. |
| MUSFNS | This workspace contains functions which when executed tell how many users are on MUSIC, how much job time you have used, etc. It also contains the function KEYBOARD that will print all the valid APL symbols. |
| BOATPROB | This workspace plays the age-old game of missionaries and cannibals, except this one makes sure you don't loose any missionaries by making a bad move. Furthermore, this game can be played in English, French or Spanish. |
| APLCOURSE | This workspace drills the user in the use of APL primitive functions and, in fact, is a good way to familiarize yourself with them. |
| TYPEDRILL | This workspace times how long it takes you to type lines of text which you supply. It can be used as a way to improve your typing speed. |
| PLOTFORMAT | This workspace contains the functions required to plot graphs on the workstation. |
| ADVANCEDEX | This workspace contains functions which can be used to find various mathematical numbers such as binomial coefficients, gcd's and it also contains functions to invert matrices. |
| WSFNS | This workspace contains functions that do the equivalent to the system functions )WIDTH, )ORIGIN, etc. |
| TEST2741 | This workspace is useful in identifying failures of a 2741 terminal and can also be used to test it after repair. This workspace is also of some value in testing other types of supported workstations. |

# APL System Commands

APL uses a set of system commands to perform the various workspace functions, as well as several other activities. These APL system commands have nothing to do with MUSIC commands which may have similar names. Each APL system command starts with a ")" character in the APL character set. (Users are cautioned that this character is probably NOT the same as that used when the workstation is not using the APL character set.)

The following are the possible APL system commands that you may use:

```
)LIB      )LOAD     )SAVE     )DROP     )COPY     )PCOPY
)CLEAR    )WSID     )WIDTH    )SI       )SIV      )ORIG
)OFF      )FNS      )VARS     )ERASE    )DIGITS
```

# MUSIC/APL I Beams

The APL language defines a set of functions known as "I Beam" functions. These functions can be used to obtain the current date and time as well as other information. The list below gives the I beam functions that you may use. Times are in sixtieths (1/60) of a second unless otherwise noted.

I19 Accumulated keyboard unlocked time in this APL session

I20 Time of day

I21 Accumulated processing time used in the APL session in units of 1/60 of a service unit

I22 Available space (bytes) remaining in current workspace

I23 Number of users signed on to MUSIC

I24 Time of day of the beginning of this APL session

I25 Date as a six digit integer of the form mmddyy

I26 The first element of vector I27

I27 The vector of statement numbers in the state indicator

# Limiting Time Option - APL

The user can limit the amount of execution time performed between reads from the workstation. This is of particular use to stop endless loops during function evaluation.

When the time limit expires, the execution of the function will be suspended and a time exceeded message will be displayed. The specific function and line number will be identified in the same manner as if the ATTN key had been hit at that moment.

```
/OPT LIMIT=nn    (nn is time limit in service units)
```

The above control statement is used to specify the limiting time value and is placed following the /LOAD APL control statement. The actual cutoff point may be slightly greater than that specified. (Note that if other /OPT statements are used, they must each appear as separate /OPT statements as it is not permissible to combine these /OPT statements into one.)

The total amount of execution time used so far can always be determined by the I-BEAM 21 function of the APL language.

# Extension of Session Time Limits - APL

The limit on the amount of execution time for a MUSIC job is normally the default job time limit associated with the user's userid. For APL, this maximum is automatically multiplied by 10 (up to a maximum of 600 service units), and a limiting time option (/OPT LIMIT=m) is set using *m* = the original time limit. This extension is not done for a batch job, or if the original time limit is 600 service units or more, or if a specific time limit is specified by /SYS TIME=n (where *n* is not MAX).

**Examples:**

1. If the user's default time limit is 8 SU and /SYS TIME= is not used, the APL job will get a limit of 80 SU and /OPT LIMIT=8 will be in effect.

2. If the user's default time limit is 180 SU and /SYS TIME= is not used, the APL job will get a limit of 600 SU and /OPT LIMIT=180 will be in effect.

3. If the user's maximum time limit is 16 SU and /SYS TIME=MAX is specified, the APL job will get a limit of 160 SU and /OPT LIMIT=16 will be in effect.

The user may specify a lower limiting time option if desired.

If the user specifies a job time limit via the /SYS TIME=n control statement (but not /SYS TIME=MAX), then the time extension will not be done nor will a limiting time option be set. This allows users to control the amount of processing time used in the usual manner.

# References - APL

*IBM APL/360 Primer* (GH20-0689)

# APL Interpreter - VSAPL

VS APL is an IBM Program Product (5748-AP1) that interprets statements written in APL. The APL language has a uniform notation, tailored to solving a great variety of problems interactively at a workstation. The language was originated to define problems concisely using well-known mathematical symbols. It supports built-in mathematical operators which permit efficient vector and matrix manipulation. It operates as both a sophisticated and powerful desk calculator and as a programming language.

VS APL supports a shared variable facility that allows APL users to communicate with the MUSIC files that may have been created by APL or other processors and editors running under MUSIC.

In VS APL, data and programs are gathered together and stored in user areas called *workspaces*. These workspaces can be stored for later use using APL commands. Workspaces created by other users can be accessed subject to the security rules of VS APL and MUSIC. A collection of public workspaces are available to assist users to perform various functions in APL.

The IBM publication *APL Language* (GC26-3847) should be used as the reference for APL.

## Usage Notes - VSAPL

VS APL runs under MUSIC in much the same way as any other program. That is, the entire APL session is regarded by MUSIC as one job.

A MUSIC execution time limit applies for your entire APL session. VS APL has a *limiting time* feature which allows user notification when excessive function evaluation time is detected. This feature is discussed in more detail later in this writeup, together with a full description of how the APL session time limit is set.

It is advisable to periodically save your current APL workspace since its contents will be lost should your workstation become disconnected from the computer for any reason.

## Running VS APL

There are two techniques to invoke VS APL. One is to type VSAPL or VSAPL.FS when the workstation is in command (*Go) mode. VSAPL.FS should only be used by users who will invoke the full screen editor functions from 3270-type workstations.

The second technique is used when you wish to specify some run options to VS APL. This is done by creating a file on the Save Library containing the sequence of commands shown below. For example, if the created file was called MYAPL, then just type the name MYAPL at *Go time to invoke VS APL with your options.

```
/SYS REGION=nnn
/FILE 1 UDS(&&TEMP)
/INCLUDE VSAPL
/OPT TRT=n
/OPT LIMIT=nnn
```

The number on the SYS command is discussed later in the topic "Workspace Concepts". The FILE command is used if the user is invoking the full-screen editor contained in the workspace called MUSIC.

The number after TRT is discussed later in the topic "APL on TTY Terminals". The number after LIMIT is discussed later in the topic "Limiting Time Option". Omit the commands that contain options you do not wish to use.

To end the APL session, the user types the APL command )OFF. This APL command will return the user to MUSIC command mode.

# Workstation Requirement - VSAPL

VS APL can be run from most workstations supported by MUSIC. However, it uses a different set of characters to those normally available. Some workstations have an optional APL character set feature. Operation of APL from workstations without the APL character set is not recommended. The following topics discuss the operation of APL on the different types of supported workstations.

## APL on 2741 and 3767 Terminals

These APL characters are available on an IBM 2741 terminal through the use of an interchangeable type element. This element is available for both the EBCD keyboard arrangement (#1167988) and the correspondence arrangement (#1167987). The last three digits of the part number is stamped on the top rim of the type element.

The APL character set for an IBM 3767 terminal is available through the alternate character set feature.

The ATTN key on these terminals is handled according to the usual APL rules. These rules are somewhat different to the usual MUSIC usage of this key.

## APL on 3270 Series Terminals

The APL character set is available on the 3270 series display terminals through the APL character set feature. To inform MUSIC that your 3270 terminal has this feature, you must use the terminal class specification of 3270A, if it is a 3277 terminal, or use 3270B, if it is a 3278 or 3279 terminal, when you sign on to MUSIC. See the /ID command for details on how to specify the trmcls parameter.

When installed, the special APL ON/OFF key is used to select the APL character set used for entry of characters. The terminal can always display APL characters. The PA1 key is used in the usual MUSIC manner to signal attention when the terminal is in WORKING mode. Escape from input in APL is through the OUT sequence signalled via the PA1 key when the terminal is in READING mode. The PA2 key is used in the usual MUSIC manner to signal that the next page is to be displayed.

## APL on TTY Terminals

Some TTY terminals have an APL characters available on them. VS APL may work correctly from these devices, though you must check this out with the one you want to use. Since various manufacturers have established different conventions in APL mode, the user must indicate which one is to be used. The TRT option described below is used to specify the scheme used. (The default value of 4 suits a large number of terminals.)

```
/OPT TRT=n    (n is 1,2,3 or 4)
```

To verify that you have chosen the right number for the TRT option, execute the function KEYBOARD contained in the MUSFNS workspace. You should get a printout like the one shown in the example at the

end of this section.

The TTY terminals should be set to transmit even parity.

Some TTY terminals can generate a few more characters than are defined for an IBM 2741 terminal. These additional characters may be rejected if an attempt is made to enter them in APL mode.

TTY terminals have no direct equivalent of the ATTN key on an IBM 2741 terminal and so the following should be done instead: Use the LINE FEED key instead of the ATTN key to correct typing errors. Use the BREAK key instead of the ATTN key in all other cases.

It is often difficult on TTY terminals to know when the system is waiting for the user to enter character data in response to a quote-quad read. The use of the /OPT BELLPR option will sound the terminal's bell when a quote-quad read is done.

# APL System Commands

APL uses a set of system commands to perform the various workspace functions, as well as several other activities. These APL system commands have nothing to do with MUSIC commands which may have similar names. Each APL system command starts with a ")" character in the APL character set. (Users are cautioned that this character is probably NOT the same as that used when the workstation is not using the APL character set.)

The following are the possible VS APL system commands that you may use:

```
)CLEAR    )CONTINUE )COPY     )DROP     )ERASE    )FNS
)GROUP    )GRP      )GRPS     )LIB      )LOAD     )OFF
)PCOPY    )QUOTA    )SAVE     )SI       )SINL     )STACK
)SYMBOLS  )VARS     )WSID     )WSSIZE
```

# Workspace Concepts - VSAPL

Your APL programs or functions are run in an area known in APL as your *workspace*. The contents of your active workspace may be saved as part of your *workspace library*. Another workspace may be loaded from this library to become the new active workspace.

The maximum size workspace you may use is determined by the size of the user region you are using. By default, MUSIC will use a region size of 108K which will result in a maximum workspace size of about 60K (K=1024 characters.) To use more than this, specify a larger region size on the MUSIC /SYS statement.

When saving a workspace, VS APL will write only the used part of current workspace to disk regardless of the size of the MUSIC region.

The APL workspaces on MUSIC are stored on the Save Library together with files created by other processors and editors. MUSIC automatically prefixes the workspace name with a special character sequence to make their names different to those of other files. For example, the workspace SAMPLE will be stored under the name @APLW.SAMPLE.

# Workspace Compatibility - VSAPL

VS APL workspaces are transferable to other VS APL installations. See your installation support personnel for assistance in this matter. APL workspaces generated by other APL versions will not work on VS APL without conversion. It may be possible to automatically convert these workspaces. The following topic discusses how to convert workspaces created on MUSIC by the older APL version.

# Workspace Conversion of old MUSIC/APL Workspaces

A conversion program exists to convert APL workspaces that were created on UDS files by the MUSIC/APL (/LOAD APL) processor. The converted workspaces will be written to the Save Library using the workspace name if possible. If the workspace name contains characters other than the standard alphanumeric ones, then the name will be converted to a suitable one. Workspace passwords, if any, will not be converted.

The conversion utility will display messages summarizing the changes made to the functions. It can run from a workstation or from batch. Running from batch may be useful when converting many workspaces.

The following sequence of commands will convert all MUSIC/APL workspaces on the UDS file described by the /FILE command. This procedure automatically requests a region size of 160K to perform the conversion.

```
/FILE 1 UDS(dsname) VOL(vvvvvv)
/INCLUDE MWSCON
   ...options if any .....
```

Use the option TEST to indicate that you do not wish the converted workspace to be written to disk. Use the option REPL option to indicate that the converted workspace is to replace any one of the same name on the user's Save Library.

# Workspace Library Numbers - VSAPL

Omit the library number when referring to workspaces stored on your own library.

Use the numbers 1, 2, 3 to refer to the public workspace libraries as described in the following topic.

Use the library number 1000 to store workspaces that are to be accessible to other users. You may wish to use a workspace password to restrict access. This is equivalent of saving a public file in the MUSIC Save Library. A )LIB command will also list workspaces saved by the user using library 1000.

Use the library number 1000 to retrieve workspaces that were saved by others using library 1000. You are not permitted to perform a )LIB 1000 command.

# APL Public Workspaces

The VS APL user on MUSIC can access several public workspaces libraries to do various functions. Library 1 contains workspaces that IBM distributes as part of VS APL. Library 3 contains other workspaces that are useful in the MUSIC environment. A list of all the workspace names can be obtained by the APL )LIB n command. To find out more about a specific one simply load it into the active workspace by the APL command )LOAD n xxx. Then type ABSTRACT, DESCRIBE or HOW to cause an abstract, description or

usage details to be displayed.

The following is a summary of the contents of the some of the public workspaces:

| | |
|---|---|
| NEWS | This workspace on library 1 contains information specific to APL users. |
| PLOT | This workspace on library 1 contains the functions required to plot graphs on the workstation. |
| EXAMPLES | This workspace on library 1 contains functions that illustrate the power of APL in solving problems. Sample functions dealing with scientific, engineering, mathematical and commercial areas are presented. |
| FORMAT | This workspace on library 1 contains functions designed to aid in the formatting of output. |
| SBIC | This workspace on library 1 contains the application illustrated in the APL Language Manual (GC26-3847). It is a skeletal system for sales, billing and inventory control. |
| MUSFNS | This workspace on library 3 contains functions which tell how much job time you have used, the current time and date, etc. It also contains the function KEYBOARD that will print all the valid APL symbols. |
| BOATPROB | This workspace on library 3 plays the age-old game of missionaries and cannibals, except this one makes sure you don't loose any missionaries by making a bad move. Furthermore, this game can be played in English, French or Spanish. |
| TEST2741 | This workspace on library 3 is useful in identifying failures of a 2741 terminal and can also be used to test it after repair. This workspace is also of some value in testing other types of supported workstations. |
| MUSIC | This workspace on library 3 contains a function that will invoke the MUSIC context editor to assist users of 3270 APL terminals modify their functions. |

# Transporting VS APL Workspaces to Other Installations

A program is available to dump (write) MUSIC VS APL workspaces to tape in a format which is transportable to VS APL installations which are not running MUSIC. The output tape is in the same format as that used by IBM when they distribute VS APL workspaces. (The FILARC program, which is described in *Chapter 10. Utilities*, should be used to send workspaces to other MUSIC installations.)

To run the program, use the following sequence of control statements:

```
/FILE 1 TAPE BLK(800) LRECL(80) VOL(xxxxxx) OLD
/INCLUDE DUMPWS
code:name1   (names of workspaces to be dumped)
code:name2
 . . .
```

The output tape must be defined by the I/O unit number 1, and must be blocked into 800 byte blocks with a logical record length of 80 bytes as shown above. Each workspace name must be specified on a separate line starting at column 1. The name must be in the form of *code*:*name*; *code* must be a valid 4 character MUSIC user code, *name* is the VS APL workspace name (maximum 11 characters). (The actual file name is "userid:@APLW.name".) The workspaces will be dumped in the order supplied and each workspace is dumped on a separate file on the tape. Since the dumped workspaces on the tape will have no workspace

names or passwords, it is advisable to keep a list of the order of the workspaces on the tape (this program produces such a list).

## Limiting Time Option - VSAPL

The user can limit the amount of execution time performed between reads from the workstation. This is of particular use to stop endless loops during function evaluation.

When the time limit expires, the execution of the function will be suspended. The specific function and line number will be identified in the same manner as if the ATTN key had been hit at that moment.

```
/OPT LIMIT=nn    (nn is time limit in service units)
```

The above control statement is used to specify the limiting time value and is placed following the /LOAD VSAPL control statement. The actual cutoff point may be slightly greater than that specified.

The total amount of execution time used so far can always be determined by the "QUAD AI" APL system variable.

## Extension of Session Time Limits - VSAPL

The limit on the amount of execution time for a MUSIC job is normally the default job time limit associated with the user's userid. For APL, this maximum is automatically multiplied by 10 (up to a maximum of 600 service units), and a limiting time option (/OPT LIMIT=m) is set using *m* = the original time limit. This extension is not done for a batch job, or if the original time limit is 600 service units or more, or if a specific time limit is specified by /SYS TIME=n (where *n* is not MAX).

**Examples:**

1. If the user's default time limit is 8 SU and /SYS TIME= is not used, the APL job will get a limit of 80 SU and /OPT LIMIT=8 will be in effect.

2. If the user's default time limit is 180 SU and /SYS TIME= is not used, the APL job will get a limit of 600 SU and /OPT LIMIT=180 will be in effect.

3. If the user's maximum time limit is 16 SU and /SYS TIME=MAX is specified, the APL job will get a limit of 160 SU and /OPT LIMIT=16 will be in effect.

The user may specify a lower limiting time option if desired.

If the user specifies a job time limit via the /SYS TIME=n control statement (but not /SYS TIME=MAX), then the time extension will not be done nor will a limiting time option be set. This allows users to control the amount of processing time used in the usual manner.

## References - VSAPL

*APL Language* (GC26-3847)

*APL/360 Primer* IBM (GH20-0689)

# Assembler - ASM

The MUSIC VS Assembler is the IBM OS/VS Assembler processor interfaced to run under MUSIC.

The VS Assembler programs must normally be assembled and executed using the OS/MUSIC interface. The /LOAD ASM procedure automatically loads this interface at execution time.

Assembler subroutines that do not perform input or output may be called from FORTRAN programs.

If more than one assembler module is to be compiled at the same time then they must be separated from each other by /OPT statements. If these /OPT statements are blank, then the options previously given will continue to be used. After loading is complete, the program is executed under control of the OS/MUSIC interface.

A program made up of only object modules produced by this Assembler and optionally COBOL object modules, can be loaded more quickly by using the /LOAD ASMLG procedure. Alternately the /LOAD LKED can be used to produce a load module from these modules.

*Notes:*

1. Since VS Assembler programs can invoke all the facilities of OS/VS operating system, some of which are not available on MUSIC, it is possible to assemble an assembler program which will not execute properly. Certain OS/VS facilities are provided by the OS/MUSIC interface. Attempts to use unavailable features results in the execution time message FEATURE NOT SUPPORTED.

2. The DXD, and CXD facilities of the VS Assembler are not supported with the /LOAD VSFORT and /LOAD LOADER procedures. Programs which are to be executed must have a CSECT or START before the first EXTRN or ENTRY. If FORTRAN and assembler programs are to be used in the same execution, the main program should be written in FORTRAN. The standard save area linkage conventions should be followed.

3. Most MUSIC system routines (such as TSTIME, NXTPGM, NPRMPT, etc,) may be called from assembler programs. For more information about these routines, consult *Chapter 9. System Subroutines* of this guide.

4. The message `FILE ERROR: CANNOT ADD SPACE TO THIS FILE` may occur for one of the assembler work files SYSUT1, SYSUT2, or SYSUT3 if your assembler program is very large. To correct this problem, supply the following /FILE statement before the /LOAD, using n=1, 2, or 3 as appropriate. The default space allocation is SPACE(200) (i.e. 200K primary space) for SYSUT1, and SPACE(150) for SYSUT2 and SYSUT3.

```
/FILE SYSUTn NAME(&&TEMP) NEW DELETE RECFM(V) SPACE(400)
```

## Macro Library - ASM

A macro library containing many OS macro-instructions is automatically available for your use (see below). The user can specify a different macro library by placing a /FILE statement defining the macro library at the beginning of the job. The /FILE statement must have a ddname of SYSLIB and the PDS parameter specified. (Consult the description of the /FILE statement for files.) For example, if the PDS parameter is specified as PDS(*.MAC), then if a macro, say, ABC is used in the program, the file ABC.MAC will be referenced. This means that all you need to do is to create a series of macros stored in files with names

XXX.MAC, where XXX is the name of the macro used. The default /FILE statement for the macro library is /FILE SYSLIB PDS(*.M). If you want to add in any special macros to the default ones, you just create files with names XXX.M as described above. You may also use /INCLUDE statements to bring in any special macros that you may write.

## Macro Instructions - ASM

The following is a list of the OS/VS macro instructions provided, showing the extent to which they are supported by the OS/MUSIC interface.

| Macro | Comments |
|---|---|
| ABEND | Completion code and dump options only. |
| CALL | Non-overlay form only. |
| CHECK | |
| CLOSE | LEAVE and REREAD only. REREAD rewinds the data set. CLOSE writes an end-of-file after the last data record. |
| DCB | All parameters, except SYNAD, supported with the following restrictions:<br>BUFNO 1 only<br>BFTEK S only<br>DSORG PS only<br>MACRF GM,GL,PM,PL,R,W only<br>RECFM F,FA,FM,FB,FBA,FBM only |
| DCBD | |
| ESTAE | Specify ABEND exit routine. |
| FREEMAIN | VC, VU, and R forms only. |
| GETMAIN | VC, VU, EC, EU, and R only. |
| GET | |
| OPEN | INPUT, OUTPUT or INOUT, OUTIN. |
| POST | |
| PUT | |
| READ | SF, length or S only. |
| RETURN | |
| SAVE | |
| SNAP | See the description later on. |
| SPIE | |
| STAE | Specify ABEND exit routine. |
| STIMER | TASK option only. The time interval must be specified as BINTVL or TUINTVL. Timer exit routines are not supported and are ignored if specified. |
| TGET | |
| TPUT | |
| TIME | |
| TTIMER | TASK option only. The time interval must be specified as BINTVL or TUINTVL. Timer exit routines are not supported and are ignored if specified. |
| WAIT | |
| WAITR | |
| WRITE | SF,length or S only. |
| WTO | Output on MUSIC unit 6. |
| WTOR | Input on MUSIC unit 9, output on MUSIC unit 6. |

# SNAP Macro - ASM

The SNAP macro instruction can be used within assembler programs on MUSIC. This instruction can produce a *snapshot* dump of the program status word (PSW), the general and floating point registers, and specified areas of main storage. The dump can be written to the workstation or to a file. SNAP can also invoke a conversational debug routine, which lets the user inspect and modify main storage, the PSW and the registers before resuming program execution. SNAP is a valuable tool for debugging assembler programs.

For a description of how to use the SNAP macro, refer to the IBM manual *OS/VS2 MVS Supervisor Services and Macro Instructions*, GC28-0683. All of the parameters described can be used on MUSIC, although some (such as TCB, SDATA and STRHDR) are ignored by MUSIC. Since the macro expands differently under MUSIC and MVS, previously written MVS programs using SNAP must be reassembled on MUSIC before they can be run on MUSIC. The execute (MF=E) and list (MF=L) forms are supported.

SNAP is not limited to programs running in OS simulation mode. It can also be used within assembler routines running in MUSIC mode. The macro generates a call to system subroutine SNAPRTN.

The following additional parameters can be used on MUSIC:

UNIT=n
: This parameter can be used instead of the DCB parameter. It specifies the logical unit number (1 to 15) to which the dump is written. For example, UNIT=6 writes the dump to the workstation.

LINE=SHORT (or LINE=S)
LINE=LONG (or LINE=L)
: This specifies whether SNAP is to use short (16 bytes per line) or long (32 bytes per line) dump lines. Short lines are usually more convenient for workstation output. The default is LINE=SHORT for workstation jobs and LINE=LONG for batch jobs.

OPT=C
: This requests that the conversational debug routine be called after the dump, before resuming program execution. The debug routine reads statements from logical unit 9 and writes to logical unit 6. OPT=C is ignored for batch jobs. To change registers or the PSW during debug, use the STORE command to change the values in main storage. The HELP command tells you where the PSW and registers are in main storage.

*Notes:*

1. When the DCB parameter is used, the SNAP routine will open the file if it is not already open. The data control block (DCB) must specify DSORG=PS, and either MACRF=PM or MACRF=W. The record format must be F, FA, FB, FBA, V, VA, VB, or VBA. Record length may be up to 160. The recommended values are MACRF=PM, RECFM=FBA, LRECL=121.

2. The PDATA parameters JPA, LPA, ALLPA, and SPLS are ignored on MUSIC.

3. The SAH parameter dumps the current save area (pointed to by register 13). The SA parameter dumps the current save area and also gives a save area traceback.

4. All registers are preserved by the SNAP routine (SNAPRTN) except R0, which is set to 0. The expansion of the SNAP macro changes R1. The register values dumped by SNAP are the values before the SNAP macro, except for R1, which has the address of the SNAP argument list.

5. The expansion of the SNAP macro usually contains OI and NI instructions (which change the condition code), so the condition code in the PSW dumped by SNAP is not meaningful.

6. For the execute form of the macro, OPT=C must be specified if wanted, even if it was specified in the

list form. Other options are changed only if specified.

**Examples:**

```
1.          SNAP  DCB=MYDCB,ID=1,PDATA=(REGS,PSW),OPT=C
                  ...
    MYDCB   DCB   DDNAME=SYSPRINT,DSORG=PS,MACRF=PM,          x
                  RECFM=FBA,LRECL=121,BLKSIZE=1210

2.          SNAP  UNIT=6,ID=2,PDATA=(REGS,PSW,SA),            x
                  STORAGE=(AREA1,END1,AREA2,END2),LINE=LONG
```

# Input/Output - ASM

Sequential input/output operations using BSAM and QSAM are supported at execution time. These I/O techniques cannot be used in combination with FORTRAN I/O in a single program.

A standard set of DDNAMES for I/O operations is provided, complete with default values for logical record length.

| DDNAME | LRECL | MUSIC I/O |
|---|---|---|
| SYSIN | 80 | input stream (data following /DATA) |
| SYSPRINT | 121 | printed output |
| SYSPUNCH | 80 | output to holding file (workstation jobs) punched output (batch jobs) |
| SYSTERM | 121 | printed output |
| CONSOLE | 80 | conversational input (workstation jobs) input stream (batch jobs) |

*Notes:*

1. Additional ddnames can be defined by specifying /FILE statements with the corresponding ddnames at the beginning of the job. (Consult the description of the /FILE statement.)

2. The default values for LRECL may be overridden by use of the LRECL= parameter in the DCB macro instruction.

3. For the WTO and WTOR macro instructions, input is performed on MUSIC unit 9 and output on MUSIC unit 6.

4. End-of-file can be indicated on a conversational read by typing /EOF.

# Usage - ASM

The Assembler is invoked by the use of a /LOAD ASM statement. This statement can be followed by a /OPT statement specifying options for the Assembler and by a /JOB statement specifying parameters for the loading step. The Assembler loads and executes the object code unless /JOB NOGO has been specified.

/LOAD ASM specifies that the lines following form a program written for the VS Assembler (and optionally, object modules), and are to be processed by the Assembler and the resulting object program is to be loaded and executed using the OS/MUSIC interface. /LOAD statements following this statement are

ignored.

## Parameters: Compile Step - ASM

```
/OPT  {NOLIST}{,NOXREF     }{,NODECK}{,SYSPARM=(xxx)}
      {LIST  }{,XREF(SHORT)}{,DECK  }
              {,XREF(FULL) }
```

| | |
|---|---|
| NOLIST | Assembled program listing is not to be printed. This is the default for jobs run from a workstation. |
| LIST | Assembled program listing is to be printed on SYSPRINT. This is the default for jobs run from batch. (If LIST is specified for a job run from a workstation, you can use the PRINT NOGEN or PRINT OFF assembler statements to reduce the amount of output to your workstation.) |
| NOXREF | No cross-reference list. This is the default for jobs run from a workstation. |
| XREF(SHORT) | A cross-reference list of all referenced statement labels is to be printed. This is the default for jobs run from batch. |
| XREF(FULL) | A cross-reference list of all statement labels is to be printed. |
| NODECK | No object module is to be produced. This is the default. |
| DECK | An object module is to be written on SYSPUNCH. |
| SYSPARM | Specifies the value to be assigned to the symbolic variable &SYSPARM. |

*Notes:*

1. Multiple /OPT statements may be used if desired. Other options that can be used are given in the IBM OS/VS Assembler Programmer's Guide publication (GC33-4021).

2. If multiple source programs are in the input stream, they must be separated from each other by /OPT statements. For this purpose, a /OPT statement with no operand may be used.

3. /FILE statements with ddnames of ASM.SYSIN, ASM.SYSPRINT, and ASM.SYSPUNCH can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions.

# Parameters: Loader Step - ASM

```
/JOB  {MAP   }{,NOGO}{,LET}{,NOPRINT}{,DUMP}{,CDUMP}{,DEBUG}
      {FULMAP}

          {,FILRFM}{,NOSEARCH}{,IOTRACE{(C)}}{,SVCTRACE{(C)}}
```

Parameters can be separated by one or more commas or blanks.

| | |
|---|---|
| MAP | List a storage map of the loaded program. |
| FULMAP | List an extended storage map of the loaded program. |
| NOGO | Do not load or execute the program. |
| LET | Allow the program to be executed even if unresolved external references (such as missing subroutines) are found. |
| NOPRINT | Informative and diagnostic messages are not to be produced. |
| DUMP | Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump. |
| CDUMP | Request a conversational trace and dump when an error occurs. |
| DEBUG | Load the DEBUG program into memory for debugging. |
| FILRFM | Supply the RECFM V if the file is V/VC. |
| NOSEARCH | Do not search the subroutine library for unresolved routines. |
| IOTRACE | Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed. |
| SVCTRACE | Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed. |

*Notes:*

1. Multiple /JOB statements may be used if desired.
2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

## Parameters: Go Step - ASM

Parameters can be passed to the GO step by specifying a "/PARM xxx" statement at the beginning of the job. *xxx* is the parameter desired.

SVC instructions and I/O operations can be traced during execution of the program. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement. The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the description of /JOB statement above.)

## Title Lines - ASM

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

## References - ASM

*IBM System/370 Principles of Operation* (GA22-7000)

*OS/VS Assembler Language* (GC33-4010)

*OS/VS Assembler Programmer's Guide* (GC33-4021)

*OS/VS2 MVS Data Management Services Guide* (GC26-3875)

*OS/VS2 MVS Data Management Macro Instructions* (GC26-3873)

*OS/VS2 MVS Supervisor Services and Macro Instructions* (GC28-0683)

## Examples - ASM

1. An assembler job with options to be used by the assembler and loader. /FILE statements for GO step (execution time) ddnames are also defined.

```
/FILE GO.CARDIN RDR
/FILE GO.PRINTER PRT
/LOAD ASM
/JOB MAP
/OPT LIST,XREF

VS Assembler source program 1

/OPT

VS Assembler source program 2

Assembler or COBOL object module(s), if any

/DATA

Data
```

2. A VS Assembler job that reads data from SYSIN (data following /DATA), lists them on SYSPRINT (printed output), and writes them on FILE01 (a temporary UDS file). When end-of-file is encountered on SYSIN, the disk data set FILE01 is rewound, and the card images on this data set are read back and listed.

```
/FILE FILE01 UDS(&&TEMP) NREC(100)
/LOAD ASM
TEST1    START  0
     SAVE   (14,12),,*        SAVE REGISTERS
         BALR   12,0              INITIALIZE BASE REGISTER
         USING  *,12              TELL ASM TO USE IT
         LR     11,13
         LA     13,SAVREG
         ST     11,4(0,13)
         ST     13,8(0,11)
         OPEN   (CARD,(INPUT),PRINT,(OUTPUT))  OPEN SYSIN +
*                                              SYSPRINT
         OPEN   (DISK,(OUTPUT))   OPEN FILE01
LOOP1    GET    CARD,WORK         READ A CARD
         PUT    PRINT,WORK        LIST IT
         PUT    DISK,WORK         WRITE IT TO DISK
         B      LOOP1             DO IT AGAIN
CDEND    CLOSE  (CARD,,DISK,REREAD)    CLOSE SYSIN,CLOSE AND
*                                      REWIND DISK FILE
         WTO    'RECORDS FROM DISK FILE'    SEND MESSAGE
         OPEN   (DISKIN,(INPUT))  RE-OPEN DISK FILE FOR INPUT
LOOP2    GET    DISKIN,WORK       READ RECORD FROM DISK
         PUT    PRINT,WORK        LIST RECORD
         B      LOOP2             DO IT AGAIN
DSKEND   CLOSE  (PRINT,,DISKIN)   CLOSE FILES
         L      13,4(0,13)        RETRIEVE SAVED REG 13
         RETURN (14,12),T         RESTORE REGS AND QUIT
SAVREG   DC     18F'0'            REGISTER SAVE AREA
WORK     DS     80C               WORK AREA
```

```
CARD     DCB    DDNAME=SYSIN,MACRF=(GM),DSORG=PS,                *
                LRECL=80,BLKSIZE=80,RECFM=F,EODAD=CDEND
PRINT    DCB    DDNAME=SYSPRINT,MACRF=(PM),DSORG=PS,             *
                LRECL=80,BLKSIZE=80,RECFM=F
DISK     DCB    DDNAME=FILE01,MACRF=(PM),DSORG=PS,               *
                LRECL=80,BLKSIZE=480,RECFM=FB
DISKIN   DCB    DDNAME=FILE01,MACRF=(GM),DSORG=PS,               *
                LRECL=80,BLKSIZE=480,RECFM=FB,EODAD=DSKEND
         END
/DATA
DATA CARD 1
DATA CARD 2
DATA CARD 3
```

# Assembler (Loader) - ASMLG

ASMLG (VS Assembler Load and Go) is the name used to access the loader. The /LOAD ASMLG statement informs MUSIC that the lines following consist exclusively of object modules created by VS Assembler and/or ANS COBOL. (This statement is functionally identical to the /LOAD COBLG statement.) Object modules will load faster if this statement is used instead of /LOAD ASM.

After loading is complete, the program is executed.

## Usage - ASMLG

This loader is invoked by the use of the /LOAD ASMLG statement. This statement can be followed by a /OPT SYSIN statement specifying the input unit number and by a /JOB statement specifying parameters for the loading step.

## Parameters - ASMLG

```
/JOB  {MAP   }{,NOGO}{,LET}{,NOPRINT}{,NOSEARCH}{,FILRFM}
      {FULMAP}
```

Parameters can be separated by one or more commas or blanks.

| | |
|---|---|
| MAP | List a storage map of the loaded program. |
| FULMAP | List an extended storage map of the loaded program. |
| NOGO | Do not load or execute the program. |
| LET | Allow the program to be executed even if unresolved external references (such as missing subroutines) are found. |
| NOPRINT | Informative and diagnostic messages are not to be produced. |
| NOSEARCH | Do not search the subroutine library for unresolved routines. |
| FILRFM | Supplies a RECFM V if the file is V/VC. |

*Notes:*

1. Multiple /JOB statements may be used if desired.
2. The DEBUG, CDUMP, DUMP, IOTRACE and SVCTRACE parameters are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the processors that do not invoke the loader directly.

```
/OPT  {SYSIN=n}
```

SYSIN=n Specifies that input is to be taken from the MUSIC unit number n. A /FILE statement defining MUSIC I/O unit n as the appropriate input file must appear at the beginning of the job. When an end-of-file or a /DATA statement is encountered on this input data set, further input is taken from the normal input stream.

## Example - ASMLG

```
/LOAD ASMLG
/JOB MAP,LET
/INCLUDE OBJ1     (VS Assembler object module)
/DATA
/INCLUDE DATA1    (user's data)
```

# IBM BASIC Environment - IBMBASIC

IBM BASIC (Program Product 5665-948) is an interactive development language. Interactive processing allows you to enter programs and data from a workstation for processing and enables you to verify results. The interactive mode also supports immediate execution of BASIC commands. Your installation may choose not to have this environment available for your use.

## Usage - IBMBASIC

Issue the command IBMBASIC to start IBM BASIC. This puts you into the full screen environment. To exit issue the QUIT command.

*Notes:*

1. IBM BASIC runs in a 3270 type environment in full screen mode.

2. The file IBMBASIC.PROFILE is used at start up time to control the BASIC environment. The user can edit this file and save a modified copy under their own userid. Refer to the *IBM BASIC Programming Guide* for details.

3. The SYSTEM command can be used to issue any MUSIC/SP command. For example

   ```
   CALL SYSTEM 'music command' or
   SYSTEM 'music command'
   ```

   can be used from a BASIC program or in immediate mode.

4. The QUERY command is not supported. The MUSIC/SP LIBRARY command can be used in its place. Use the SYSTEM command to issue the MUSIC/SP LIBRARY command. For example

   ```
   SYSTEM 'LIBRARY *TEST*.BASIC'
   ```

   lists all files that contain TEST in the file name that are BASIC source.

5. Since IBM BASIC programs can invoke facilities of the MVS operating system, some of these are not available on MUSIC/SP. Refer to the *IBM BASIC/MVS Systems Services* manual for more details.

6. The IBM BASIC/GDDM interface is not supported under MUSIC/SP.

7. The IBM BASIC/SQL interface is not supported under MUSIC/SP.

8. File names are limited to 8 characters if not enclosed in quotes. BASIC automatically appends a qualifier to the end of file names not enclosed in quotes. The BASIC qualifiers are:

   | | |
   |---|---|
   | .BASDATA | data files, |
   | .BASIC | source files, |
   | .BASLOG | log files, |
   | .BASOBJ | workspace files, |
   | .PROFILE | profile files, |
   | .LIST | list files, |
   | .OBJ | object files. |

A quoted file name is not automatically qualified.
So, a MUSIC/SP file name can be used
if the file name is enclosed in quotes.
The following two statements are examples of the use of
file names that would be qualified by BASIC.
BASIC would load MYFILE.BASIC and would open MYFILE.BASDATA
respectively.

```
load MYFILE
open #1: 'MYFILE',OUTPUT,NATIVE,SEQUENTIAL,FIXED
```

The following two statements are examples of the use of file names that would be used as is by BASIC. BASIC would load MYFILE and would open MYFILE.OUTPUT respectively.

```
load 'MYFILE'
open #1: "'MYFILE.OUTPUT'",OUTPUT,NATIVE,SEQUENTIAL,FIXED
```

The /FILE definition can be used if the entered file name is the same as a DDNAME that has been setup on a /FILE statement. To define your /FILE definitions to the BASIC environment, make a copy of the file $IBB:IBMBASIC and save it on your userid. Edit it and add your /FILE statements and save it.

9. BASIC automatically creates fixed compressed (FC) files on MUSIC/SP for the following types of files: BASDATA, BASIC, BASOBJ, PROFILE, and OBJ with a record length of 80; BASLOG, LIST with a record length of 133; and for all other FIXED format files with a record length of 80.

   BASIC creates variable compressed (VC) files when you use the VARIABLE option on the BASIC OPEN statement.

   You must create a file outside of BASIC if you need to create a file that is different from the defaults. You can use the following system call to create a file

   ```
   system 'CREATE.BASFILE N(MYTEST) LRECL(250) NEW(REPL)'
   ```

   This call creates the file MYTEST with a record length of 250.

   The program CREATE.BASFILE is the IBM BASIC version of the MUSIC/SP /FILE statement. This allows you to specify any /FILE statement options on the system call to CREATE.BASFILE. After you have defined your file using the system call, you can access this file in your BASIC program via its file name.

   Alternatively, you can use the /FILE statement to define a file to be used within the BASIC environment. Make a copy of the file $IBB:IBMBASIC and save it on your userid. Edit it and add your /FILE statements and save it.

10. A direct access file must be created initially with dummy records. The program CREATE.DIRECT can be used to create your direct access file. CREATE.DIRECT prompts you for the file name, number of records, and record size. Then it creates the file and fills it with dummy records. Then the direct access file can be used from within your BASIC program. You can use the following system call to create a direct access file from within BASIC

    ```
    system 'CREATE.DIRECT'
    ```

11. The execution of OBJECT files created by IBM BASIC is supported when running within or outside of the BASIC environment. Use the COMPILE command to compile your BASIC program and store the object deck in a file. Use the RUN command to run the object deck within the IBM BASIC environment. For example

```
RUN MYPROG (OBJ
```

runs MYPROG from object module. You can execute the OBJECT file by using the MUSIC/SP linkage editor outside of the IBM BASIC environment.

The performance of using compiled object files within the Interactive BASIC environment is slow. Running from a load module provides better performance.

# IBM BASIC programs as load modules

An IBM BASIC program can be run outside of the IBM BASIC environment as a load module. The following example shows the steps to create and run a load module. In this example, the object module for the BASIC program resides in the file MYBASIC.OBJ.

1. Link editing the object module MYBASIC.OBJ to create a load module

```
/FILE LMOD N(MYBASIC.LMOD) NEW(REPL)
/LOAD LKED
/JOB NOGO,MODE=OS
/INC MYBASIC.OBJ
/INC $IBB:IBMBASIC.OBJ
/INC $IBB:MBLIOMRT.OBJ
 ENTRY BLIOMRT
 NAME MYBASIC
```

2. Executing the program MYBASIC

```
/SYS REG=400
/FILE LMOD N(MYBASIC.LMOD)
/LOAD XMON
MYBASIC
```

# IBM BASIC Interlanguage Communications

IBM BASIC programs can invoke routines written in Assembler, COBOL, FORTRAN, and PLI. The following example calls the MUSIC/SP subroutine TSUSER. As defined by IBM BASIC interlanguage communications, you must call FLINK with the FORTRAN routine name first before you call the FORTRAN routine.

```
100 INTEGER N
105 CALL FLINK ('TSUSER')
110 CALL FORTRAN ('TSUSER',1,N)
120 PRINT 'The value returned from TSUSER was ',N
130 END
```

There are a few things to keep in mind when you are using VS FORTRAN subroutines with your BASIC program.

1. BASIC character variables have two lengths, the current length and the maximum length. The current length must be assigned to the variable before the variable is used in the calling sequence for the FORTRAN routine. This can be done by assigning spaces to the whole length of the variable. For example

```
DIM A$*8
A$ = '        '
CALL FLINK ('MYFORT')
CALL FORTRAN ('MYFORT',A$)
```

2. If you are doing any I/O within your program and you call a FORTRAN routine, you must define FT05F001 and FT06F001 to IBMBASIC before you start. Make a copy of the file $IBB:IBMBASIC and save it on your userid. Edit it and add the following two /FILE statements and save it.

```
/FILE FT05F001 RDR
/FILE FT06F001 PRT
```

On MUSIC/SP, when you run a program, a load module is created and executed. To use routines in other languages, you must include these object decks in the link edit that BASIC does behind the scenes. The program that does these steps for you is $IBB:@RUNTIME.LOADLIB. You should make a copy of this file on your userid and add /INC statements for your object modules at link edit time.

Refer to the *IBM BASIC Programming Guide* for further details on interlanguage communications.

# IBM BASIC VSAM Support

IBM BASIC supports VSAM under MUSIC/SP. You must first create your VSAM file using the MUSIC/SP utility program AMS. See Chapter 10 of this manual for further details on AMS. In order to use VSAM, you must start BASIC with a /FILE statement for the VSAM file. Make a copy of the file $IBB:IBMBASIC and save it on your userid. Edit it and add a /FILE statement for the VSAM file. Refer to the *IBM BASIC Programming Guide* for further details on how to use VSAM with IBM BASIC.

# References - IBMBASIC

*B is for BASIC* (GS26-4102)

*IBM BASIC General Information* (GC26-4023)

*IBM BASIC Language Reference Summary* (SX26-3736)

*IBM BASIC Language Reference* (GS26-4026)

*IBM BASIC Programming Guide* (SC26-4027)

*IBM BASIC/MVS Systems Services* (SC26-4106)

# BASIC Compiler - VSBASIC

The VS BASIC compiler is an optional IBM Program Product (5748-XX1) that may be available for your use. VS BASIC programs may be entered and modified using the usual MUSIC techniques or you may use the MUSIC VS BASIC Editor. This editor is designed specifically to handle VS BASIC programs. The writeup for this special editor can be found immediately following this writeup about the VS BASIC Compiler. The standard MUSIC editor may also be used for entering and modifying VS BASIC programs and data.

## VS BASIC Highlights

- Arithmetic facilities, in short and long precision, enabling the user to assign values and to perform arithmetic operations on variables and arrays.
- Character facilities, enabling the user to define character variables of different lengths, to concatenate groups of character data items, and to locate and extract substrings within character strings.
- Array facilities, enabling the user to define one and two dimensional character and arithmetic arrays, to re-dimension arrays, to assign a single value to all elements of an array, and to sort arrays into ascending or descending order.
- Stream-oriented file facilities, enabling the user to read and write files, and to reposition files to their beginning or their end.
- Record-oriented input/output functions for both sequential and direct access files.
- Output formatting capabilities, enabling the user to position data items on a print line, and offering commercial formats with comma insertions, asterisk protection, and floating plus, minus, and dollar signs.
- A comprehensive library of built-in mathematical and character functions, performing numerous arithmetic, character-string, and conversion operations.
- User-defined functions, enabling the user to define special-purpose functions not available as built-in functions. Such functions can be defined in one statement or over many statements, and can specify multiple, single, or no arguments.
- Program segmentation statements, enabling the user to sequentially execute a number of separate BASIC programs to fit special needs, and to pass information between these programs.
- Internal constants, supplying commonly-used arithmetic values such as pi and the square root of 2, and providing such metric conversions as inches to centimeters and gallons to liters.

## Language Specifications - VSBASIC

The language specifications may be found in the IBM publication *SYSTEM/370 VS BASIC Language* (GC28-8303).

# Usage Notes - VSBASIC

- The DELETE FILE, KEY, NOKEY, DUPKEY, NOREC, and DUPREC statements for record-oriented input/output are not supported. This means that they may be coded but will have no effect. REWRITE is only supported for direct files.

- Direct files, those using the REC clause, must be files with a record format of FIXED (F). Note that the system default is FIXED COMPRESSED (FC). FIXED record format files may be created using /FILE statements. For example,

```
/FILE 1 NAME(XXX) NEW LRECL(100) RECFM(F) SPACE(50) NORLSE
/LOAD IEFBR
```

  will create a FIXED format file called XXX with a record length of 100 and space of 50K.

- Record-oriented files must be allocated prior to running the VS BASIC program.

- MUSIC checks the file's record format to determine which type of access is valid. If it is FIXED, the file is assumed to be direct (RRDS), and both sequential and direct access are allowed. For files with any other record formats, or UDS files, the file is assumed to be sequential (ESDS).

- After a FIXED file has been accessed by VS BASIC, the MUSIC /LIST command, and other related commands, may get confused as to the number of lines in the file, since they have not been processed sequentially. Editing the file and saving it back will reset the end of file pointers and cause this problem to disappear.

- The MUSIC subroutine library is not available to VS BASIC programs, and VS BASIC cannot communicate with subroutines written in other languages.

- The EOF clause on the PUT statement is not supported for files (ddnames) SYSIN, SYSPUNCH, SYSPRINT and CONSOLE, unless they are specified on /FILE statements at the beginning of the job.

- When the CHAIN statement is used, the file name is taken to be the name of a file. If the file name of "*" is given then the chained program is assumed to be the next one in sequence in the input stream.

- It is possible to chain to programs written in other languages. To do this, the user must use a slightly modified version of the CHAIN command as follows:

```
CHAIN '%name parameters'
```

  If the first character of the name field is specified as a "%" sign, the program contained in the following file name is executed. Any characters following the file name are passed to the program as parameters.

**Examples:**

```
250 CHAIN '%FILE1'      execute the program in FILE1
100 CHAIN '%PROG X286' execute the program in PROG.  The string 'X286' is passed as a
                       parameter.
```

- The upward pointing arrow on TTY terminals is treated as if it were the same as the vertical bar (|) character. Therefore use the symbol ** for exponentiation in VS BASIC.

- To conserve main storage, REMARK statements are not passed to the compiler, unless the REM compiler option is specified. Therefore, if REMARK statements are referenced by other statements

such as GOTO, the REM option must be used.

- Source statements are limited to 80 columns in length. See the following discussion of continuation statements for one method of getting around this limitation.

## Continuation Statements - VSBASIC

Since VS BASIC source statements on MUSIC are limited to 80 columns, it is occasionally necessary to continue a statement onto another line. This is done by using a line number of the form *n.m* on the continuation lines. *n* is the number of the BASIC statement and m (1 or more) is the continuation line number. A maximum of two continuation lines may be used.

The compiler concatenates the continuation text to the end of the previous line (at column 80). The continuation text is considered to start immediately after the *n.m* line number if the following character is non-blank, or one character after the *n.m* otherwise.

**Examples:**

```
120 LET X = A + B
120.10 + C + D
120.20 + E + F

200 INPUT A,B,C,
200.1 D,E,F
```

## Main Storage Requirements - VSBASIC

The amount of main storage available for the user's program (object code plus array storage) depends upon the amount of buffer space being used and the space required for storing the source statements, but is usually about 25K (K=1024 bytes) in a 108K user region. This limits the size of each VS BASIC program to about 200 to 250 statements. However, several programs may be chained together by use of the CHAIN statement. Alternately, use a "/SYS REGION=nnn" statement to ask for a larger user region. Also, if the VS BASIC compiler and library are kept resident in the system link pack areas (this is done by your MUSIC installation, not by you), then the space available is increased by about 50K.

## Control Statement Set Up - VSBASIC

The MUSIC VS BASIC Editor is available to assist in typing in, making corrections and running a BASIC program. This facility is described in more detail in the separate writeup that follows this one. Alternately a VS BASIC program can be entered and run in a method similar to the other MUSIC processors as follows:

```
/FILE statements (if necessary)
/LOAD VSBASIC
/OPT parameters  (see below).
...VS BASIC source program or object module.
...
/DATA
...data
...
```

*Notes:*

1. On the /LOAD statement, VSBASIC may be abbreviated as VSBAS.

2. Any number of /OPT statements may be used.

3. A parameter specified on a /OPT statement remains in effect until specifically changed by a later /OPT statement.

4. More than one source or object program may be used. They are compiled and executed separately, one after another. Each must be separated from the preceding program by one or more /OPT statements. For this purpose it is permissible to use a /OPT statement which does not contain any parameters. A program may use the CHAIN statement to pass information to the next program.

5. If more than one program is present, only the last one is able to read from the input stream (data following the /DATA statement). However, DATA statements can be used within each source program.

6. Following the /LOAD VSBASIC statement, any control line (other than /OPT, /JOB or /DATA) containing a slash in column 1 is ignored. A /JOB statement is treated exactly like /OPT.

7. The user may type /EOF to indicate end of file on a conversational read.

# VS BASIC Object Modules

The VS BASIC compiler can produce an object module representing the BASIC program, and this object module can later be used in place of the source program. However, this module is not a standard object module, and cannot be processed by /LOAD LOADER. For most VS BASIC programs, there is no advantage in using an object module, since the time to compile the source is usually less than the time to load the object module.

# VS BASIC Compiler Parameters - /OPT Statement

Parameters on the /OPT control statement are used to specify compiler options, input/output options, and other information. Parameters must be separated by commas and must not contain embedded blanks. One blank must be present between /OPT and the first parameter.

| | |
|---|---|
| LIST<br>NOLIST | Specifies whether or not a listing of the source program is to be printed. Default is NOLIST for workstations and LIST for batch. |
| PRINT<br>NOPRINT | NOPRINT suppresses the compile and execution time messages. |
| GO<br>NOGO | NOGO means that the program is to be compiled but not executed. The default is GO. |
| SPREC<br>LPREC | SPREC specifies that the program is to use short precision for arithmetic calculations. LPREC specifies long precision. The default is SPREC. SPREC and LPREC are ignored if specified for an object module, since the option in effect at the time the program was compiled is always used. |

SYSIN=n This specifies the unit number n from which additional compiler input is to be read. The additional input may contain source, object, and control statements. When end of file or a /DATA statement is reached on the specified unit, reading continues with the lines following the /OPT statement containing the SYSIN=n parameter.

INPUT=n Specifies the input unit number to be used for reads resulting from execution of INPUT statements in the BASIC program. Unit 9 (conversational input) is assumed if this parameter is not used.

REM
NOREM To conserve main storage, REMARK statements are usually not passed to the compiler. Use the REM option if you wish REMARK statements to be processed. If remark statements are referenced by other statements, (such as GOTO), then you must use the REM option. The default is NOREM.

BUF=n Specifies the number of bytes to be reserved for data set buffers. The default is 4096 bytes. If several data sets are used, a value greater than 4096 may result in more efficient input/output. However, specifying a value greater than 4096 decreases the space available for object code and arrays. If the BUF parameter is used, it should appear on the first /OPT statement.

BUFF=n Same as BUF=n.

NOPRINT This parameter causes some information messages to be suppressed, for example the messages giving compile and execution times.

LINESIZE=n Specifies the maximum line length to be used for workstation display during execution of the BASIC program. The default is 120, and the allowable range is 72 to 132.

DECK
NODECK Specifies whether or not an object module is to be produced. The default is NODECK. If DECK is specified, the object module will be written to SYSPUNCH (see the description of ddnames below).

STORE
NOSTORE Same as DECK and NODECK.

Examples of /OPT statements:

```
/OPT LPREC,LIST

/OPT NOPRINT,LINESIZE=100,BUF=8192

/OPT NOGO
```

# Control of Input/Output Files - VSBASIC

User files referenced by VS BASIC input/output statements are identified by a file name specified on the BASIC statement. This file name is associated with a MUSIC User Data Set or file in the following way:

1. VS BASIC will first attempt to match the file name with the ddnames defined on /FILE statements at the beginning of the job. A file name referenced this way must be 1 to 8 characters long.

2. If the file name cannot be matched, VS BASIC will try to resolve it by looking for a file with the specified file name in the Save Library. File names referenced this way can be 1 to 17 characters in

length.

3. For stream-oriented files, if the file cannot be found in the Save Library but is to be opened for output, a new file will be created with the specified file name.

# Pre-Defined DDNAMES - VSBASIC

| ddname | Default LRECL | Maximum LRECL | Minimum LRECL | MUSIC I/O |
|---|---|---|---|---|
| SYSIN | 80 | 80 | 20 | input stream (data following /DATA) |
| SYSPRINT | 133 | 133 | 20 | printed output |
| SYSPUNCH | 80 | 80 | 20 | holding file (workstation jobs)<br>or punched output (batch) |
| CONSOLE | 80 | 80 | 20 | conversational input (workstation jobs)<br>or input stream (batch) |

The pre-defined ddnames can be overridden by specifying /FILE statements with the corresponding ddnames at the beginning of the job.

# Examples of VS BASIC Programs

**Example 1**

```
*Go
list vsbsm2
*In progress
/LOAD VSBASIC
10 REM PROGRAM TO CALCULATE COMPOUND INTEREST
20 PRINT ' '
30 PRINT 'COMPOUND INTEREST PROGRAM...'
40 PRINT ' '
50 PRINT 'ENTER INITIAL AMOUNT'
60 INPUT P
70 PRINT 'ENTER INTEREST RATE (E.G. 4.5)'
80 INPUT I
90 PRINT 'ENTER NO. OF PERIODS PER YEAR'
100 INPUT N
110 PRINT 'ENTER NO. OF YEARS'
120 INPUT Y
130 FOR J=1 TO N*Y
140 P=P+P*I/(100*N)
150 NEXT J
160 PRINT 'FINAL AMOUNT IS',P
170 GO TO 50
180 END
*End
*Go
vsbsm2
*In progress
VS BASIC
COMPILE =   0.12 SEC
```

```
COMPOUND INTEREST PROGRAM...

ENTER INITIAL AMOUNT
?
200.00
ENTER INTEREST RATE (E.G. 4.5)
?
9.5
ENTER NO. OF PERIODS PER YEAR
?
4
ENTER NO. OF YEARS
?
3
FINAL AMOUNT IS   265.0674
ENTER INITIAL AMOUNT
?
/eof
** END OF FILE ON UNIT   9
EXEC =   0.17 SEC
*End
*Go
```

**Example 2**

```
*Go
list vsbsm3
*In progress
/FILE FILE01 UDS(&&TEMP) NREC(100)
/LOAD VSBASIC
10 REM  THIS PROGRAM READS NUMBERS FROM UNIT 9,
20 REM  WRITES THEM TO THE TEMPORARY UDS FILE01,
30 REM  THEN READS THEM BACK AND PRINTS THEM.
40 INPUT A,B,C
50 IF A=999 GO TO 80
60 PUT 'FILE01',A,B,c
70 GO TO 40
80 CLOSE 'FILE01'
90 GET 'FILE01',A,B,C, EOF 120
100 PRINT A,B,C
110 GO TO 90
120 END
*End
*Go
vsbsm3
*In progress
VS BASIC
COMPILE =   0.12 SEC
?
1.2, 3.4, 5.6
?
10,-20,30
?
999,0,0
     1.2       3.4       5.6
     10        -20       30
EXEC =   0.16 SEC
*End
*Go
```

**Example 3**

```
*Go
list vsbsm5
*In progress
/LOAD VSBASIC
10 REM THIS PROGRAM READS IN NUMBERS FROM A FILE
20 REM AND DISPLAYS THEM ON THE WORKSTATION.
30 PRINT 'ENTER FILE NAME'
40 INPUT A$
50 GET A$,X,EOF 80
60 PRINT X
70 GOTO 50
80 END
*End
*Go
vsbsm5
*In progress
VS BASIC
COMPILE =   0.12 SEC
ENTER SAVE FILE NAME
dat77
 10
 20
 30
EXEC =   0.13 SEC
*End
*Go
```

**Example 4. Copying one sequential file to another.**

```
10 DIM A$80
20 READ FILE 'FILE1', A$, EOF 50
30 WRITE FILE 'FILE2', A$
40 GO TO 20
50 PRINT ' END OF COPY'
60 END
```

**Example 5. Printing selected records from a direct file.**

```
10 DIM A$80
20 PRINT ' ENTER NUMBER OF RECORD TO PRINT'
30 INPUT I
40 IF I=0 GO TO 80
50 READ FILE 'DATAFILE', REC=I, A$
60 PRINT A$
70 GO TO 20
80 END
```

# VS BASIC Editor

This VS BASIC Editor provides a convenient method of entering, modifying and running programs written in the VS BASIC Language. It realizes the fact that all BASIC statements have unique line numbers assigned in numerical order.

This editor is a standard feature of the MUSIC system support for the optional IBM VS BASIC compiler.

The normal MUSIC editor (the /EDIT command) may also be used for entering and modifying VS BASIC programs and data files. The major advantages of the VS BASIC Editor are that it is statement number oriented and can renumber BASIC programs.

BASIC statements may be entered in any order, since the editor will automatically arrange them in order of their BASIC line number. Should two or more statements have the same line number, the one entered last will be taken. An entire statement entered previously may be deleted simply by typing in just the BASIC statement number with nothing following it.

Special commands are provided for list, delete, change, save, execute, and renumber operations. These commands refer to lines of the file by their line numbers. Line numbers are of the following form:

nnnnn

or nnnnn.mmm

where *nnnnn* is the 1 to 5-digit BASIC statement number, and *mmm* is an optional 1 to 3-digit continuation statement number. Blanks may optionally be used preceding or following the line number, but the line number itself must not contain embedded blanks.

## Types of Input Lines - VSBASIC Editor

The following types of input lines are accepted:

1. Control statements with / in column 1:

   These are accepted only at the beginning of the file. They are automatically assigned line numbers 0.10, 0.20, 0.30, etc. and may be referenced by these line numbers for purposes of replacement, listing, insertion, etc. If no / statements are present, the line '/LOAD VSBASIC' is automatically supplied as the first line of the file.

2. BASIC source statements:

   nnnnn text

   or nnnnn.mmm text

3. Special commands, optionally identified by $ in column 1:

   These commands (described below) enable the user to list, change or delete lines of the file, renumber the BASIC program, turn statement number prompting on or off, save and execute the updated file, and perform various other operations. The commands consist of a command keyword followed by a list of operands. A blank is optional following the command keyword, except for the SAVE, EXEC and FILE commands, where a blank is required. Normally the operands are separated by commas, but one or more blanks (zero or more in the case of CHANGE) may be used instead of a comma.

   A command may be identified as a command by typing a dollar sign ($) before the command name, for example: $CHANGE, $SAVE. The $ character is optional except when line number prompting is active.

## Invoking the VS BASIC Editor

The VS BASIC Editor is invoked by the MUSIC command /BASIC, which is used as follows:

```
/BASIC xxxxxx
```

where *xxxxxx* is the name of an existing file which is to be edited. If the name is omitted, then the contents of the existing /INPUT file will be used. When the name NEW or NULFIL is used, the edit begins with an empty file; this is useful when a new program is to be typed in.

The file *xxxxxx* to be edited must contain only MUSIC control statements (at the beginning of the file) and the BASIC source statements for a single BASIC program. The file must not contain a /DATA statement followed by lines of data, but the BASIC DATA statement may be used. Files which do not conform to these restrictions should be edited using the MUSIC Editor.

# VS BASIC Editor Commands

```
CHANGE n,/string1/string2/
C
```

This command changes *string1* to *string2* in the specified line n. The new line is displayed. LAST may be specified instead of a line number, meaning the last line of the file. If the line number is omitted, the change is applied to the last line typed in by the user. *string2* may be null (that is, of zero length) but not *string1*. Normally a slash ("/") is used for the string delimiter, but any character other than a digit, letter (A-Z), blank, comma or period may be used instead. The third delimiter may be omitted.

```
DELETE  {n1}{,n2}
DEL
```

This command is similar to LIST, except that the specified line or lines are deleted from the file. A line may also be deleted by simply typing its line number.

```
END
```

This command writes the updated file to unit 10 and terminates the editor. The user must then type a '/SAVE name,SV' or '/SV name' command in order to save the file.

```
EXEC  {name}
EX
```

This command terminates the editor, saves the updated file under the specified name (using a scheduled "/SV name" command), and then executes the BASIC program (using a scheduled "/EXEC name"

command). If no name is specified on the command, then the original file name is used. If there is a possibility that the scheduled /SV command will not be successful, this command should not be used. (If the automatic SAVE does not work, then the changed version of the file will be lost.)

```
FILE  {name}
```

This command saves the updated file, using the specified file name or the name of the original file if no name is specified on the command. The editor is then terminated. If the name NULFIL or NEW was used on the /BASIC command, the user will be prompted to enter a file name.

```
LIST  {n1}{,n2}
L
```

The LIST command displays the specified line or range of lines, or the entire file if no line numbers are specified. LAST may be specified instead of a line number, and means the last line of the file.

```
PRINT  {n1}{,n2}
P
```

The PRINT command is the same as LIST.

```
PROMPT {n,m   }
PR     {OFF   }
       {x     }
       {NOCHAR}
```

The PROMPT command is used either to request statement number prompting or to specify a prompt character.

When n and m are used, the editor begins issuing statement number prompts, using n as the starting number and m as the increment. The increment m is optional, defaulting to the previous m, or to 10 on the first $PROMPT command. If OFF is specified, statement number prompting is terminated. n and m may be from 1 to 99999. Initially prompting is off.

When the PROMPT command is used without any operand, it is taken as "PROMPT n" where *n* is the smallest multiple of the current increment which exceeds the largest line number in the file. This automatically allows lines to be added to the end of the file.

When statement number prompting is in effect, the user may type the text of the statement, a $ command, or a line beginning with an overriding statement number.

When *x* is specified on the command, it is used as a conversational read prompt when statement number prompting is not done. *x* must be a single character other than a letter or digit. The NOCHAR option removes the prompt character. A conversational read prompt is useful for workstations whose keyboards

do not lock.

```
QUIT
```

This command terminates the editor without saving the file.

```
RENUM a,b,c,d
REN
```

This command renumbers the BASIC program. The renumbering process includes scanning for statement number references and altering them appropriately. It is assumed that the program conforms to the language and syntax rules of VS BASIC. It is recommended that the program be compiled (in order to detect and correct language errors) before being renumbered. The parameters are:

a Starting new line number to be used. The default is 10.
b The statement number increment to be used. The default is 10.
c The old statement number of the first line of the range of lines to be renumbered. If this parameter is omitted, renumbering begins at the beginning of the file.
d The old statement number of the last line of the range of lines to be renumbered. If this parameter is omitted, renumbering continues until the end of the file.

```
RUN
RU
```

This command is similar to EXEC, except that the MUSIC input file (/INPUT) is automatically used. Thus the new file replaces the /INPUT file and then the BASIC program is executed from the /INPUT file.

```
SAVE  {name}
SV
```

The SAVE command causes the current updated version of the file being edited to replace the file that has the specified file name, without terminating the editor. This is similar to the FILE command except that the edit session will not be terminated. If no name is specified on the command, then the replacement will be done on the original file being edited. The SAVE command cannot be used to store into the /INPUT file.

## Examples of VS BASIC Editor Commands

```
LIST 10,300

LIST 5,20.2

L LAST
```

```
DEL50,50.1

CHANGE 25,/X=1.9/X=0.19/

C25#ABC#DE#

C/LTE/LET/
```

## Sample Usage of VS BASIC Editor

The following session illustrates how the VS BASIC Editor may be used to enter a BASIC program, make changes to it, save it, and execute it. Lines typed by the user are shown in lower case; lines displayed at the workstation are shown in upper case.

(Some users find that it is more convenient to use the /INPUT file to temporarily hold a copy of the BASIC program until they have debugged it. This can be done by replacing the /BASIC NUFIL command in the following by /BASIC and by using a RUN command instead of the EXEC FILEPQ given here.)

```
/basic new
*In progress
BASIC
10 rem this program adds 2 numbers
20 print 'enter 2 numbers to be aded
25 input a,b
30 c=a+b
40 go to 25
50 end
35 print 'the sum is',c
c 20 @aded@added'@
20 PRINT 'ENTER 2 NUMBERS TO BE ADDED'
renum
RENUMBERED
list
/LOAD VSBASIC
10 REM THIS PROGRAM ADDS 2 NUMBERS
20 PRINT 'ENTER 2 NUMBERS TO BE ADDED'
30 INPUT A,B
40 C=A+B
50 PRINT 'THE SUM IS',C
60 GO TO 30
70 END
exec filepq
FILED
VS BASIC
COMPILE =   0.12 SEC
ENTER 2 NUMBERS TO BE ADDED
?
1.234, 2.005
THE SUM IS      3.239
?
/eof
**END OF FILE ON UNIT   9
EXEC =   0.06 SEC
*End
*Go
```

# IBM C/370 Compilers

Programs written in the C language can be compiled and run using one of two IBM C compilers: IBM C/370 compiler (Program Product 5688-040) or the IBM C/370 Compiler Version 2 (5688-187) and IBM C/370 Compiler Version 2 Library (5688-188).

The C processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed.

## Usage - C/370

The C/370 Compiler is invoked by the use of a /LOAD C370 statement. This processor invokes the C/370 Compiler first, and then the loader, upon successful completion of the compilation process. After loading, the compiled program is automatically executed using the OS/MUSIC interface (unless /JOB NOGO is used or the compiler return code is 12 or more). The /OPT and /JOB control statements may be used with this processor. C/370 object modules can optionally be produced and saved, intermixed with source. The object modules are identical with those produced on OS.

## Available Unit Numbers and Buffer Space - C/370

Any tape blocksize up to 32760 may be used, provided the user region size (specified on /SYS REGION=nnn) is large enough.

The user can use 3 UDS files. MUSIC I/O unit 4 cannot be defined in /FILE statements since it is used for the compiler modules.

## External File Names - C/370

A standard set of external file names (ddnames) is provided, along with default values for logical record length (LRECL).

| DDNAME | LRECL | MUSIC I/O |
|---|---|---|
| SYSIN | 80 | input stream (data following /DATA) |
| SYSPRINT | 121 | printed output |
| SYSLIN | 80 | output to holding file (workstation jobs)<br>punched output (batch jobs) |
| SYSTERM | 121 | printed output |
| SUBLIB | - | object modules for C/370 library functions |
| SYSLIB | 80 | PDS containing C/370 header files (.H in source) |
| SYSMSGE | - | PDS containing C/370 compiler messages |
| SYSCPRT | 133 | print output (source listing) |
| SYSUT1 | 80 | compiler work file |
| SYSUT2 | 80 | compiler work file |
| SYSUT3 | 80 | compiler work file |
| SYSUT4 | 80 | compiler work file |
| SYSUT5 | 80 | compiler work file |
| SYSUT6 | 3200 | compiler work file |
| SYSUT7 | 3200 | compiler work file |
| SYSUT8 | 3200 | compiler work file |
| SYSUT9 | 137 | compiler work file |
| SYSUT10 | 133 | compiler work file |

Additional ddnames can be defined by specifying /FILE statements with the corresponding ddnames at the beginning of the job. (Consult the description of the /FILE statement.)

Note that in some cases, file SYSCPRT must refer to a dataset and not to a workstation or printer. For example, when the LIST parameter is specified, C/370 attempts to read the SYSCPRT file. A workstation or printer cannot be rewound and read, and so the compile step fails.

In order to specify user-defined header files, use the following MUSIC JCL statement:

```
/FILE SYSLIB PDS(*.HDR,$IBC:C#3.*.HDR)
```

End-of-file can be indicated on a conversational read by typing /EOF.

# Parameters: Compiler Step - C/370

Compiler options may be specified on a /OPT statement preceding the C/370 source. The options should be separated by commas, and the last option must be followed by a blank. A /OPT statement may also be used to indicate the start of a new external procedure. If more than one /OPT statement is used, the effect is cumulative, that is, each option remains in effect for the rest of the job until reset by a later /OPT statement.

The C/370 '#pragma options' statement within the source file may be used in place of a /OPT statement for some options. Refer to the definition of the #pragma statement in the C/370 documentation for a list of those options that may be specified. Options on #pragma statements are not cumulative. Note that options specified on a /OPT statement override #pragma options specified in source.

The default compiler options are listed below. Refer to the *IBM C/370 User's Guide* (SC09-1264-01) for a description of the options and their abbreviations.

| Workstation Defaults: | Batch Defaults (if different): |
|---|---|
| NOAGGREGATE | |
| NOALIAS | |
| DECK | |
| NOEXPMAC | |
| NOFLAG | |
| NOGONUM | |
| LANGLVL(EXTENDED) | |
| NOLIST | |
| MARGINS(1,72) | |
| NOOFFSET | |
| NOOPTIMIZE | |
| NOPPONLY | |
| NORENT | |
| SEQUENCE(73,80) | |
| NOSOURCE | SOURCE |
| TARGET() | |
| TERMINAL | |
| NOTEST(SYM,BLOCK,LINE) | |
| NOUPCONV | |
| NOXREF | |

/FILE statements with ddnames of C370.SYSIN, C370.SYSCPRT, and C370.SYSLIN can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions. To provide more space to any of the default compiler work files SYSUTn ("n" can be 1 through 10; the default work space is 100K) for very large programs, supply the following /FILE statement before /LOAD C370:

```
/FILE SYSUTn NAME(&&TEMP) RECFM(F) LRECL(mmm) SPACE(550) NEW DELETE
```

where *mmm* is the logical record length as given in the table above.

## Parameters: Loader Step - C/370

```
/JOB  {MAP    }{,NOGO}{,LET}{,NOPRINT}{,DUMP}{,CDUMP}{,DEBUG}
      {FULMAP}

         {,FILRFM}{,NOSEARCH}{,IOTRACE{(C)}}{,SVCTRACE{(C)}}
```

Parameters can be separated by one or more commas or blanks.

MAP List a storage map of the loaded program.

FULMAP List an extended storage map of the loaded program.

NOGO Do not load or execute the program.

LET Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.

NOPRINT Informative and diagnostic messages are not to be produced.

DUMP Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump.

CDUMP Request a conversational trace and dump when an error occurs.

DEBUG Load the DEBUG program into memory for debugging.

FILRFM Supply the RECFM V if the file is V/VC.

NOSEARCH Do not search the subroutine library for unresolved routines.

IOTRACE Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed.

SVCTRACE Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed.

*Notes:*

1. Multiple /JOB statements may be used if desired.

2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. Other OS-mode processors that do not invoke the OS-mode loader directly.

## Parameters: Go Step - C/370

Execution-time options can be specified by using a #pragma runopts statement in the source program, or by specifying a "/PARM xxx" statement at the beginning of the job, where *xxx* is the option desired. However, in order for the C/370 runtime routines to recognize this, it is necessary to specify the following statement in your source program:

```
#pragma runopts(execops)
```

since by default no execution-time options are recognized on the /PARM statement.

Regardless of whether execution-time options are specified, arguments to the application program must start with a leading '/'. If this is not present, then no arguments will be passed to the application program. Note that the leading '/' is not passed to the application program.

The /PARM statement should be coded as follows. If no execution-time options are required, the leading '/' is still required if application program arguments are to be supplied to the application program. If there are no application program arguments, the '/' is not required.

```
/PARM execution-time options / application-arguments
```

The default execution-time options are:

HEAP(4K,4K,ANYWHERE,KEEP)
ISAINC(0)
ISASIZE(0)
LANGUAGE(UENGLISH)
NOREPORT
TEST(NONE,*,;)
STAE
SPIE

SVC instructions and I/O operations can be traced at execution time. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the /JOB statement for a description of the syntax.)

## Title Lines - C/370

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

## References - C/370

*IBM C/370 User's Guide*, (SC09-1264).

*Systems Application Architecture Common Programming Interface C Reference - Level 2*, (SC09-1308).

*IBM C Reference Summary*, (SX09-1088).

*IBM C/370 Diagnosis Guide and Reference V2*, (LY09-1804).

## Example - C/370

This famous program writes a message to the workstation.

```
/LOAD C370

  /* This is a sample C/370 Program */

#include <stdio.h>
main ()
{
  printf("Hello, World\n");
}
```

# CICS/VM-MUSIC Interface - CICS

MUSIC/SP interfaces directly to CICS/VM (Customer Information Control System/VM) Release 2 (product number 5684-011) running under VM. Translation and preparation of the application source files are performed under MUSIC/SP. When the application is to be tested, an automatic transfer of the required files is performed to one of a pool of CMS accounts, and the application is started. When finished, your MUSIC/SP session resumes.

This facility allows you to develop CICS applications under MUSIC/SP, and then perform your testing as required under CMS without requiring you to own a CMS account.

## Usage - CICS

You start the CICS/VM-MUSIC interface by issuing the command "CICS". This brings up the panel displayed in Figure 8.3.

```
------------------------ VM/CICS Interface ------------------ Files:
Command ==>

Create NEW Entry Name:       Type:     (Aapp,Capp,Cdat,Cmap,Papp,Alst)
   Execute Appl. Name:         (Application LIST to be executed)

 _ ACCT00.CAPP
 _ ACCT01.CAPP            Selection Options
 _ ACCT02.CAPP
 _ ACCT03.CAPP              D - Delete entry
 _ ACCT04.CAPP              E - Edit entry
 _ ACCTSET.CMAP             G - Generate Application & Screen MAP
                            L - Create LIST of application & screens
                            T - Translate and compile CICS source
                            V - View entry
                            X - Execute selected application & screens
                                Select the items and then press PF4.


------------------------------------------------------------------------
PFKeys 1-Help 2-Setup 3-Exit 4-Exec 5-Loc 7-Up  8-Dn 10-Refresh 12-Cmd
```

*Figure 8.3 - CICS Interface*

You can create CICS applications in Cobol, PL/I or Assembler, utilizing the command level Application Programming Interface (API) of CICS. This is commonly known as the EXEC CICS interface. Maps, created with BMS statements are also created with this facility.

Programs can be translated, which replaces the EXEC CICS commands with the appropriate language CALLs and then compiles the program to create an object file. The translator will detect API statements that are not supported and notify the user. Applications containing these statements will not compile.

Basic Mapping Support (BMS) is an interface between CICS and its application programs. BMS commands have a simple generalized form, because formatting information is stored separately in what are called maps.

Each map has two forms, *physical* and *symbolic*.

BMS formats a display by embedding control characters in the data stream. A *physical map* tells it how to do this.

A *symbolic description map* is a source language data structure that the assembler or compiler uses to resolve source program references to fields in the map.

The *command area* allows for the execution of MUSIC commands. In addition, the following commands are handled directly by the VM/CICS interface.

| | |
|---|---|
| Top | Move to the beginning of the list |
| Last/Bottom | Move to the bottom of the list |
| Locate | Find the next occurrence of 'string' |

To *create* a new application program file or map, the name of the file and its type must be specified. The name is a 1-8 character name that begins with the letter (A-Z). The remaining characters may be letters or numbers.

The *type* must be one of the following:

AAPP - File contains an Assembler application program
CAPP - File contains a COBOL application program
CDAT - File contains date to be used by the CICS program
PAPP - File contains a PL/I application program
CMAP - File contains BMS macro statements, defining a map
ALST - Name defines an application list that can be executed under the CICS/VM product

To execute an application under the CICS/VM product, the name of an application list must be specified. When selected, the specified application files and maps are accessible to the CICS/VM product running under the VM/CMS environment. When the CICS/VM environment completes control is returned to the CICS Interface.

# Application Lists - CICS

CICS applications and Maps can be operated on in a variety of ways from the selection screen. A list of existing applications and maps is presented. To the left of the list is a command option, which may be specified as one of the following:

## Options - CICS

D Delete the item from the system. Once deleted, it can NOT be accessed.

E Edit the item. You may edit applications, maps or application lists.

G Create the physical and/or symbolic maps. The file must contain BMS macro statements. The type of map created is specified via the SETUP screen. By default BOTH the physical and symbolic maps are generated.

L Specify which application programs and maps are grouped together to form an application list. The application list can then be specified for execution under the CICS/VM product.

To create a list, place "L" on each item that will be place of the list. You may move between multiple screens. The "L" will be changed to an "*" each time a function key or ENTER key is pressed. Once all the individual items have been selected, the name of the new application list is specified via the "Create NEW Application Name" field. The type must be selected as "alst".

T Translate the EXEC CICS API into source language statements. If the translator completes without any errors, the appropriate compiler/assembler is invoked to create an object file. The file must be a CICS application.

V View the item. You may view applications, maps or application lists.

X Specify which application programs and maps are grouped together and then executed under the CICS/VM product. This option is similar to the "L", but an application list is NOT created. This allows for testing of specific applications before including them in an application list.

Place "X" on each item that will be grouped together for execution. You may move between multiple screens. The "X" will be changed to an "*" each time a function key or ENTER key is pressed. Once all the individual items have been selected, press the F4 key to execute under the CICS/VM product.

## Program Function Keys - CICS

F2 Select the VM/CICS Setup facility. Options allow for the specification of translator and compiler options. Refer to the help associated with the SETUP facility for details.

F3 Exit the VM/CICS Interface

F4 Execute the items that have been selected via the X option. You must make the selections prior to pressing this key.

F5 Find the next occurrence of the string specified via the last LOCATE command.

F7 Scroll UP the list of applications programs, lists and maps.

F8 Scroll DOWN the list of application programs, lists and maps.

F10 Refresh the list of application programs, lists and maps, by scanning the library. Normally the list is automatically updated when a new items is created. This automatic updating can be turned-off via a SETUP option.

F12 Retrieve the last command issued from the command area to allow for re-executing or changing the command.

# CICS/VM Setup Facility

The Setup screen sets the options used by the VM/CICS interface to translate and compile applications and maps. Make any changes and press ENTER to effect the change.

Changes made to the options can be either on a temporary basis (just for this session) or permanently (retained for future sessions).

Refresh file list — By Default (Y) VM/CICS updates the list of applications and maps after adding a new item. If N is specified, the list will NOT automatically be updated after an add.

Options for BMS macros to produce a DSECT and symbolic map.

Generation Option — Use this option to specify the type of map output. The choices are:

| | |
|---|---|
| BOTH | - produce the physical map and dsect (default) |
| MAP | - produce the physical map only |
| DSECT | - produce the DSECT only |

Syslib Search — Specify the search order used by the Assembler to locate the BMS macros used in the created of the Dsect and symbolic maps. The default is $CIC:*.AMAC,*.M. You would normally add additional search lists at the end.

The Source Translator translates the EXEC CICS commands in a CICS/VM program and produces two output files:

- A file containing the translated output, ready for compilation.
- A file containing the translator listing.

Translator Options — Specify the translator options. These are as described in the "CICS/VS Application Programmer's Reference". Note, however, that you must use the abbreviated forms of any options having more than 8 characters. For example, you must specify OS (instead of OPSEQUENCE). Each option is separated by a blank.

```
------------------------ VM/CICS Interface Setup ---------------------

                                  Refresh File List after ADD:   (Y/N)
MAP
   Generation Option:          (Both, Csect, Dsect)
 Syslib Search Order:

Source Translator
  Translator Options:

COBOL
       Compiler type:          (COBOL or COBOL2)
     Compile Options:
 Syslib Search Order:

ASSEMBLER
   Assembler Options:
 Syslib Search Order:

PL/I
     Compile Options:
 Syslib Search Order:
=============================================================================
PF-Keys:  1-Help  3-Exit  5-Temporary Change   12-Exit NO Change
```

*Figure 8.4 - CICS Interface Setup*

Options for the COBOL Compiler are:

Compiler Type — Specify which Cobol Compiler you are using. COBOL2 (the default), specifies that the VS COBOL II compiler be used. COBOL will utilize the OS/VS COBOL

compiler.

Compiler Options — Specify the COBOL Compiler options to be used when compiling the program. Each option is separated by a comma.

Syslib Search — Specify the search order used by the Compiler to locate the files when the COPY or BASIS verbs are used. The default is $CIC:*.CMAC,*.COPY,*.CUSERMAC. You would normally add additional search lists at the end.

Options for the Assembler are:

Assemble Options — Specify the Assembler options to be used when assembling the program. Each option is separated by a comma.

Syslib Search — Specify the search order used by the Assembler to locate the macros used in the program. The default is $CIC:*.AMAC,*.M,*.AUSERMAC. You would normally add additional search lists at the end.

Options for the PL/I Compiler are:

PL/I Options — Specify the PL/I compiler options to be used when compiling the program. Each option is separated by a comma.

Syslib Search — Specify the search order used by the Compiler to locate the %INCLUDE information used in the program. The default is $CIC:*.PMAC,*.M,*.PUSERMAC. You would normally add additional search lists at the end.

## Program Function Keys - CICS

F3 — Save the changes made and return to the main selection screen. The changes are permanently and will be used for all further operations.

F5 — Temporarily save the changes and return to the selection screen. The changes will ONLY be in effect for this session. The next time the VM/CICS interface is started, the permanently saved options will be used.

F12 — Return to the selection screen without changing any options.

# References - CICS

*CICS/VM Application Programming Guide*, (SC33-0570).

*CICS/VM General Information*, (GC33-0571).

# COBOL Compiler - COBOL2

MUSIC supports the OS/VS COBOL II (Program Product 5668-958). The COBOL2 processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed.

Since COBOL II programs can invoke many of the facilities of the Operating System, some of which are not available on MUSIC, it is possible to compile a COBOL II program which will not execute on MUSIC.

## Usage Notes - COBOL2

1. Labeled (standard or nonstandard) magnetic tapes are not recognized as such on MUSIC/SP. If the labels are present on tape, then they will appear as a separate file at the beginning of the tape.

2. The library management and teleprocessing (TCAM) features are not supported.

3. The AIXBLD run-time option is not supported. Dynamic building of VSAM alternate indexes is not available. The alternate index must be created using Access Methods Services (see AMS in *Chapter 10. Utilities*).

4. The MERGE verb is not supported.

5. The SORT control dataset is not used and the use of the SORT-CONTROL special register is not supported and is ignored. The following special sort control registers may be used: SORT-CORE-SIZE, SORT-RETURN, and SORT-MESSAGE. The RERUN clause is not supported.

6. Using QSAM (files whose organization is sequential), the following restrictions apply:

   - Open Extend is not supported. The use of the 'APPEND' on the /FILE statement will provide the same result.
   - File status codes of 34 and 39 are not supported.
   - The I-O option on the OPEN statement is not supported.

7. The following restrictions apply to the COBOL II interactive debugger:

   - COBOL II Debug is supported in line mode only.
   - The ONABEND command is not supported and is ignored.
   - A COBOL II program that calls GDDM functions cannot be debugged.
   - If the COBOL II program being debugged abends due to a program interruption, the COBOL statement in error will be displayed on the workstation and the program and debug environment will be terminated. Control will not return to the debug environment.

## Input/Output - COBOL2

Input/output operations using QSAM and VSAM are supported. Thus sequential, keyed, and direct access techniques are available on MUSIC/SP using VSAM, and sequential access is also available using QSAM. These operations require the use of the OS/MUSIC interface. This interface is automatically loaded with the /LOAD COBOL procedure. When object modules are used alone you should use the /LOAD COBLG procedure to load them.

For files whose organization is sequential, this I-O option on the OPEN statement is not supported, and the REWRITE statements cannot be used.

# System Names - COBOL2

In COBOL II, the programmer establishes a correspondence between file names defined in the program and external input/output devices by means of a SELECT clause in the FILE-CONTROL section. For example, the COBOL II statement:

```
SELECT CARD-FILE ASSIGN TO UR-2540R-S-SYSIN
```

assigns the file name CARD-FILE to the input stream. The I/O description UR-2540R-S-SYSIN is called the system name.

This system name is made up of four components, the *Device Class* (UR), the *Device Number* (2540R), the *File Organization* (S), and the *external Name* (SYSIN). MUSIC supports the following subset of System Name specifications:

| | |
|---|---|
| Device Class | UR, UT, or DA |
| Device Number | 2540R, 2540P, 1403, 2400, 2314, or 3330 |
| File Organization | S, AS |
| External Name | SYSIN, SYSPUNCH, SYSPRINT, SYSTERM, SYSOUT, CONSOLE, SYSDBOUT (or a user-specified name - see Note 1 below) |

# External Names - COBOL2

A standard set of external names (DDNAMES) is provided, complete with default values for logical record length and blocksize.

| EXTERNAL NAME | LRECL | MUSIC I/O |
|---|---|---|
| SYSIN | 80 | input stream (data following /DATA) |
| SYSPRINT | 121 | printed output |
| SYSPUNCH | 80 | output to holding file (workstation job)<br>punched output (batch job) |
| SYSTERM | 121 | printed output |
| SYSOUT | 121 | printed output |
| SYSDBOUT | 121 | printed output |
| CONSOLE | 80 | conversational input (workstation jobs)<br>input stream (batch jobs) |

*Notes:*

1. Additional external names can be defined by specifying /FILE statements with the corresponding external names (ddnames) at the beginning of the job. (Consult the description of the /FILE statement.)

2. The default values for LRECL and BLKSIZE are obtained by use of the RECORD CONTAINS and BLOCK CONTAINS clauses. In other areas, the blocksize and the logical record length are defined by the file description entry.

3. DISPLAY may be used alone or with the UPON CONSOLE or UPON SYSPRINT options. ACCEPT may be used with FROM CONSOLE or FROM SYSIN. For batch jobs, ACCEPT FROM CONSOLE

is equivalent to ACCEPT FROM SYSIN.

4. When a disk output data set is closed, an end-of-file is written at the end of the data set. A COBOL CLOSE followed by an OPEN has the effect of rewinding the file. Data sets except for disk data sets, should always be processed using the LABEL RECORDS ARE OMITTED clause.

5. End-of-file can be indicated on a conversational read by typing /EOF.

6. To override the default work files (SYSUT1 to SYSUT7) for providing more space, define a /FILE statement as the following and place it before /LOAD COBOL:

```
/FILE SYSUTn NAME(&&TEMP) RECFM(V) NEW DELETE SPACE(sss)
```

   where *n* is 1,2,3,4,5,6 or 7 and *sss* is the number of K-bytes wanted. The default is SPACE(100).

7. For unresolved externals you should point to the COBOL 2 subroutine library. Do this by including:

```
/FILE SUBLIB PDS(*COBOL2,*OS)
                                (for load and go)
/FILE SUBLIBOS PDS(*COBOL2,*OS)
                                (for linkage editor)
```

   Those references will be resolved then.

8. The following statement is needed when you use /LOAD LKED to execute a COBOL2 load module:

```
/FILE SUBLIBOS PDS(*COBOL2,*MUS,*OS,*EXT)
```

# Usage - COBOL2

The COBOL compiler is invoked by the use of a /LOAD COBOL2 statement. This statement can be followed by a /OPT statement specifying options for the Assembler and by a /JOB statement specifying parameters for the loading step. The COBOL compiler loads and executes the object code unless /JOB NOGO has been specified.

/LOAD COBOL2 specifies that the lines following are a COBOL II program (and optionally, object modules), and are to be processed by the VS COBOL II Program Product compiler and the resulting object program is to be loaded and executed using the OS/MUSIC interface. /LOAD statements following this statement are ignored. Previously compiled object modules may be included in the input stream.

An input stream consisting entirely of COBOL II object modules (and optionally the VS Assembler object modules) can be loaded more quickly for execution by specifying /LOAD COBLG or /LOAD ASMLG to invoke the OS-mode loader. Alternately the /LOAD LKED can be used to produce a load module from these modules.

Multiple COBOL programs (for example, a main program and subprograms) may appear in a single input stream. Each source program must be separated by a /OPT statement.

For information on the use of VSAM files with COBOL, refer to the topic "MUSIC/SP VSAM" in *Chapter 4. File System and I/O Interface* of this guide.

Sort facilities are provided as described under "Using COBOL's SORT Feature" in *Chapter 10. Utilities* of this guide.

If the COBOL COPY or BASIS verb is used, a /FILE statement with a ddname of SYSLIB and parameter PDS must be defined at the beginning of the job. (Consult the description of the /FILE statement for files.) The MUSIC /INCLUDE statement may also be used to perform a function similar to the COBOL COPY verb.

## Parameters: Compile Step - COBOL2

```
/OPT  [parms...]   (see below for possible parameters)
```

The specified parameters should be separated by commas, and the last parameter must be followed by a blank. A partial list of parameters and their MUSIC defaults is shown here. Additional parameters may be found in the *IBM COBOL II Programmer's Guide*.

BUF= The amount of main storage allocated to buffers. The default value is BUF=16000, which will allow small and medium size source programs to be compiled relatively quickly. If the message INSUFFICIENT CORE FOR THIS JOB is printed, the user may reduce this value (for example, /OPT BUF=6000).

NOSOURCE Program source statements not to be listed. This is the default for jobs run at the workstation.

SOURCE Program source statements to be printed on SYSPRINT. This is the default for jobs run from batch.

NODECK No object module to be produced. This is the default.

DECK An object module is to be written on SYSPUNCH.

NOSEQ The compiler is not to do sequence checking on the source program statements.

SEQ The compiler is to do sequence checking on the source program statements. This is the default.

*Notes:*

1. Multiple /OPT statements may be used if desired. For other parameters which may be used on the /OPT statement, refer to the IBM COBOL Programmer's Guide publication.

2. /FILE statements with ddnames of COB.SYSIN, COB.SYSPRINT, and COB.SYSPUNCH can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions.

# Parameters: Loader Step

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,DUMP][,CDUMP][,DEBUG]
      [FULMAP]

          [,FILRFM][,NOSEARCH][,IOTRACE[(C)]][,SVCTRACE[(C)]]
```

Parameters can be separated by one or more commas or blanks.

| | |
|---|---|
| MAP | List a storage map of the loaded program. |
| FULMAP | List an extended storage map of the loaded program. |
| NOGO | Do not load or execute the program. |
| LET | Allow the program to be executed even if unresolved external references (such as missing subroutines) are found. |
| NOPRINT | Informative and diagnostic messages are not to be produced. |
| DUMP | Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump. |
| CDUMP | Request a conversational trace and dump when an error occurs. |
| DEBUG | Load the DEBUG program into memory for debugging. |
| FILRFM | Supply the RECFM V if the file is V/VC. |
| NOSEARCH | Do not search the subroutine library for unresolved routines. |
| IOTRACE | Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed. |
| SVCTRACE | Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed. |

*Notes:*

1. Multiple /JOB statements may be used if desired.
2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

# Parameters: Go Step - COBOL2

Parameters can be passed to the GO step by specifying a "/PARM xxx" statement at the beginning of the job. *xxx* is the parameter desired.

SVC instructions and I/O operations can be traced during execution of the program. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement. The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the description of /JOB statement above.)

# Title Lines - COBOL2

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

# COBOL II Interactive Debugger

COBTEST, the VS COBOL II debug tool, is a flexible tool for monitoring the execution of the VS COBOL II program. With COBTEST you can suspend program execution, continue execution, skip sections of code, correct errors, display and set variables, set up expected input, and display output.

After filling in the panel fields, the system will:

1. Compile the specified source program.
2. Create an executable program (load module)
3. Start the COBOL program, within the debug environment

In Figure 8.5 the initial panel for COBTEST is displayed.

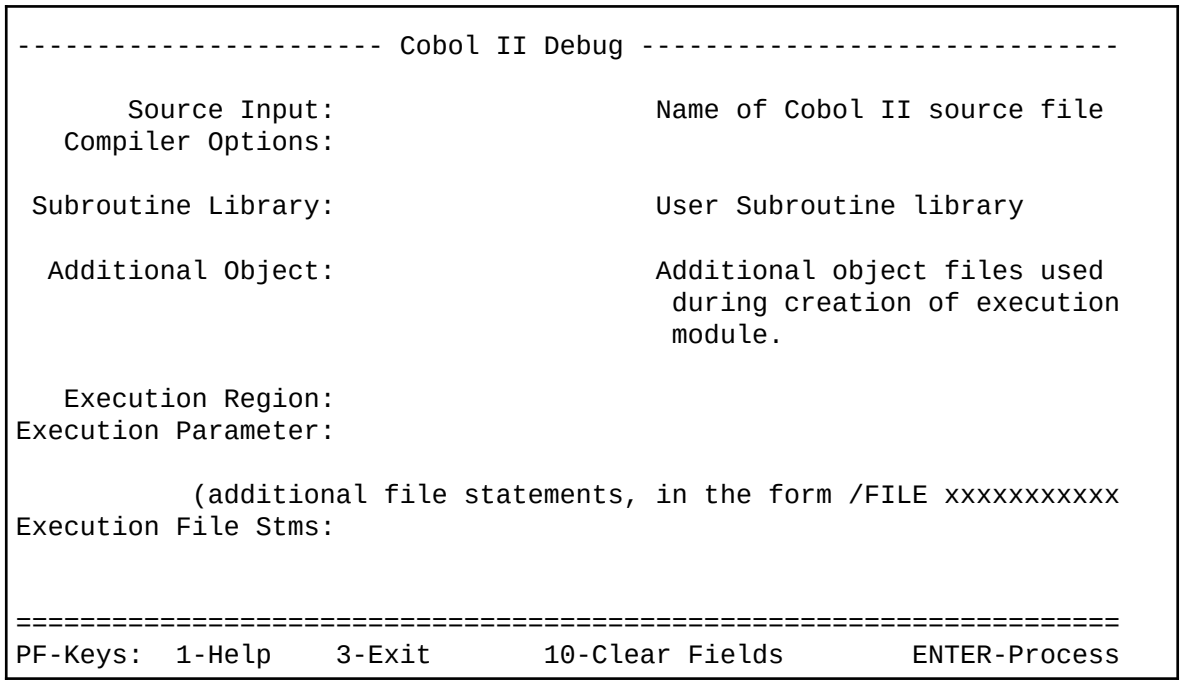



*Figure 8.5 - COBOL II Debug*

Source Input — Specifies the name of the file containing the COBOL II Source statements that will be compiled and executed under the COBOL II interactive debug environment. This is a required field.

Compiler Options — Specify any COBOL II compiler options that you wish to use during the compile phase. The default options are: Apost, Lib, Source, Deck, Res, Test.

Subroutine Library — Specify the name of **your** subroutine library that you wish to use during the link edit process.

Additional Object — Specify the file names of any other files that will be used during the link edit process. The files **must** be object type files.

Execution Region — Specify the size of the User region to be used during the execution of the program. The default is 800 K.

Execution Parameters — Specify any parameters that your program is expecting at program initiation. You may also specify run-time parameters for the COBOL II environment. The format of this field is: user parameters / run-time parameters.

Execution File Statement — Specify any FILE statements that your program requires during program execution. The complete FILE statement needs to be specified.

By default, information that is written to SYSDBOUT will be placed in the file @@DBOUT. This can be overridden by specifying a FILE statement in the Execution File Statement field.

## Keys - COBOL2

3 Terminate the facility. No compilation or program execution will be done.

10 Clear all of the screen fields to the default. All fields are blank, except for execution region which is 800K.

Enter Process the information specified in the different fields and compile, link edit and execute the program under the COBOL II Debug environment.

# References - COBOL2

*IBM VS COBOL for OS/VS* (GC26-3857).

*OS/VS COBOL Compiler and Library Programmer's Guide* (SC28-6483).

# Examples - COBOL2

1. A COBOL job with compiler and loader options. /FILE statements for GO step (execution time) ddnames are also defined.

```
/FILE GO.CARDIN RDR
/FILE GO.PRINTOUT PRT
/LOAD COBOL2
/OPT DECK
/JOB MAP
      COBOL Source Program
/OPT
      COBOL Source Subprogram
/DATA
      Data
```

2. A sample COBOL program which scans a file.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    SAMPLE.
       AUTHOR.        IBM PROGRAMMER.
       INSTALLATION.  STL
       DATE-WRITTEN.  MAY 25, 1987.
       DATE-COMPILED.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-370.
       OBJECT-COMPUTER.  IBM-370.
       INPUT-OUTPUT SECTION.
         FILE-CONTROL.
            SELECT       INVENTORY
            ASSIGN       INVENT
            FILE STATUS  INVENT-STATUS.
       DATA DIVISION.
       FILE SECTION.
           BLOCK CONTAINS 0 RECORDS
           RECORD 80 CHARACTERS
           LABEL RECORDS STANDARD
           DATA RECORD INVENTORY-RECORD.
       01  INVENTORY-RECORD          PIC X(80).
```

```
   WORKING-STORAGE SECTION.
   77  E-O-F-FLAG                PIC X   VALUE 'N'.
       88  INVENT-E-O-F                  VALUE 'A'.
   77  INVENT-STATUS             PIC XX.
   PROCEDURE DIVISION.
   0000-MAIN-LINE.
        OPEN INPUT INVENTORY
          IF INVENT-STATUS NOT EQUAL ZERO
            THEN DISPLAY 'OPEN ERROR ON '
-             'FILE INVENTORY' UPON CONSOLE
            GOBACK
          END-IF
        INITIALIZE INVENTORY-RECORD
                   INVENT-STATUS.
   FD  INVENTORY
       RECORDING MODE F
       BLOCK CONTAINS 0 RECORDS
       RECORD 80 CHARACTERS
       LABEL RECORDS STANDARD
       DATA RECORD INVENTORY-RECORD.
   01  INVENTORY-RECORD          PIC X(80).
   WORKING-STORAGE SECTION.
   77  E-O-F-FLAG                PIC X   VALUE 'N'.
       88  INVENT-E-O-F                  VALUE 'A'.
   77  INVENT-STATUS             PIC XX.
   PROCEDURE DIVISION.
   0000-MAIN-LINE.
        OPEN INPUT INVENTORY
          IF INVENT-STATUS NOT EQUAL ZERO
            THEN DISPLAY 'OPEN ERROR ON '
-             'FILE INVENTORY' UPON CONSOLE
            GOBACK
          END-IF
        INITIALIZE INVENTORY-RECORD
                   INVENT-STATUS.
        PERFORM TEST BEFORE
                UNTIL INVENT-E-O-F
          READ INVENTORY
            AT END
               SET INVENT-E-O-F           TO TRUE
          END-READ
          IF INVENT-STATUS GREATER THAN 10
            THEN DISPLAY 'READ ERROR ON '
-             'FILE INVENTORY' UPON CONSOLE
            SET INVENT-E-O-F             TO TRUE
          END-IF
          DISPLAY INVENTORY-RECORD
        END-PERFORM
        CLOSE INVENTORY
        GOBACK.
```

# COBOL Compiler - VSCOBOL

MUSIC supports the OS/VS COBOL (Program Product 5740-CB1). The COBOL processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed with the OS/MUSIC interface.

*Notes:*

1. Since COBOL programs can invoke many of the facilities of the Operating System, some of which are not available on MUSIC, it is possible to compile an COBOL program which will not execute on MUSIC.

2. MUSIC supports fixed length sequential COBOL files. Programs using indexed, partitioned or direct access techniques may be compiled but not run under MUSIC. Direct access using the relative record technique is supported - see "Direct Access Support" below.

3. For information on the use of VSAM files with COBOL, refer to the topic "MUSIC/SP VSAM" in *Chapter 4. File System and I/O Interface* of this guide.

4. Sort facilities are provided as described under "Using COBOL's SORT Feature" in *Chapter 10. Utilities* of this guide.

5. If the COBOL COPY or BASIS verb is used, a /FILE statement with a ddname of SYSLIB and parameter PDS must be defined at the beginning of the job. (Consult the description of the /FILE statement for files.) The MUSIC /INCLUDE statement may also be used to perform a function similar to the COBOL COPY verb.

6. Labeled (standard or nonstandard) magnetic tapes are not recognized as such on MUSIC. If the labels are present on tape, then they will appear as a separate file at the beginning of the tape.

7. The library management, debug and teleprocessing (TCAM) features are not supported.

## Input/Output - VSCOBOL

Sequential input/output operations using BSAM and QSAM are supported. These operations require the use of the OS/MUSIC interface. This interface is automatically loaded with the /LOAD COBOL procedure. When object modules are used alone you should use the /LOAD COBLG procedure to load them.

## Direct Access Support - VSCOBOL

For COBOL programs to be executed under MUSIC, MUSIC supports the *relative file* organization. This is a direct access technique in which the records of the file are numbered 0,1,2,... This number is called the *nominal key*. Records can be read (by the READ statement) and updated (by the REWRITE statement) randomly by specifying the nominal key. A relative file may also be read (READ statement) and written (WRITE statement) sequentially; in fact, they are normally created using sequential writes.

This requires special care on MUSIC, because sequential access is done using the BSAM access method and random access is done using the BDAM access method, and these two access methods store records differently on disk. BSAM and BDAM storage methods coincide only if the MUSIC record format is F and the

MUSIC record length is a multiple of 512.

For this reason, the /FILE statement defining a COBOL relative record file which is being created sequentially must specify RECFM(F) and LRECL(n), where n is the record length used in the COBOL program, rounded up to a multiple of 512. For example, if the program uses a record length of 100, then the MUSIC file or UDS file must be created as RECFM(F) LRECL(512).

## System Names - VSCOBOL

In COBOL, the programmer establishes a correspondence between file names defined in the program and external input/output devices by means of a SELECT clause in the FILE-CONTROL section. For example, the COBOL statement:

```
SELECT CARD-FILE ASSIGN TO UR-2540R-S-SYSIN
```

assigns the file name CARD-FILE to the input stream. The I/O description UR-2540R-S-SYSIN is called the system name.

This system name is made up of four components, the *Device Class* (UR), the *Device Number* (2540R), the *File Organization* (S), and the *external Name* (SYSIN). MUSIC supports the following subset of System Name specifications:

| | |
|---|---|
| Device Class | UR, UT, or DA |
| Device Number | 2540R, 2540P, 1403, 2400, 2314, or 3330 |
| File Organization | S |
| External Name | SYSIN, SYSPUNCH, SYSPRINT, SYSTERM, SYSOUT, CONSOLE (or a user-specified name - see Note 1 below) |

## External Names - VSCOBOL

A standard set of external names (DDNAMES) is provided, complete with default values for logical record length and blocksize.

| EXTERNAL NAME | LRECL | MUSIC I/O |
|---|---|---|
| SYSIN | 80 | input stream (data following /DATA) |
| SYSPRINT | 121 | printed output |
| SYSPUNCH | 80 | output to holding file (workstation job)<br>punched output (batch job) |
| SYSTERM | 121 | printed output |
| SYSOUT | 121 | printed output |
| CONSOLE | 80 | conversational input (workstation jobs)<br>input stream (batch jobs) |

*Notes:*

1. Additional external names can be defined by specifying /FILE statements with the corresponding external names (ddnames) at the beginning of the job. (Consult the description of the /FILE statement.)

2. The default values for LRECL and BLKSIZE are obtained by use of the RECORD CONTAINS and BLOCK CONTAINS clauses. In other areas, the blocksize and the logical record length are defined by

the file description entry.

3. DISPLAY may be used alone or with the UPON CONSOLE or UPON SYSPRINT options. ACCEPT may be used with FROM CONSOLE or FROM SYSIN. For batch jobs, ACCEPT FROM CONSOLE is equivalent to ACCEPT FROM SYSIN.

4. When a disk output data set is closed, an end-of-file is written at the end of the data set. A COBOL CLOSE followed by an OPEN has the effect of rewinding the file. Data sets except for disk data sets, should always be processed using the LABEL RECORDS ARE OMITTED clause.

5. End-of-file can be indicated on a conversational read by typing /EOF.

6. To override the default work files (SYSUT1 to SYSUT5) for providing more space, define a /FILE statement as the following and place it before /LOAD COBOL:

```
/FILE SYSUTn NAME(&&TEMP) RECFM(V) NEW DELETE SPACE(sss)
```

   where *n* is 1,2,3,4, or 5 and *sss* is the number of K-bytes wanted. The default is SPACE(100).

7. The use of the FIPS flagger option in VS COBOL requires a SYSUT6 file. This may be defined by placing a /FILE statement, such as the following, before /LOAD COBOL:

```
/FILE SYSUT6 NAME(&&TEMP) RECFM(V) NEW DELETE
```

## Usage - VSCOBOL

The COBOL compiler is invoked by the use of a /LOAD COBOL statement. This statement can be followed by a /OPT statement specifying options for the Assembler and by a /JOB statement specifying parameters for the loading step. The COBOL compiler loads and executes the object code unless /JOB NOGO has been specified.

/LOAD COBOL specifies that the lines following are an COBOL program (and optionally, object modules), and are to be processed by the COBOL Program Product compiler and the resulting object program is to be loaded and executed using the OS/MUSIC interface. /LOAD statements following this statement are ignored. Previously compiled object modules may be included in the input stream.

An input stream consisting entirely of COBOL object modules (and optionally the VS Assembler object modules) can be loaded more quickly for execution by specifying /LOAD COBLG or /LOAD ASMLG to invoke the OS-mode loader. Alternately the /LOAD LKED can be used to produce a load module from these modules.

Multiple COBOL programs (for example, a main program and subprograms) may appear in a single input stream. Each source program must be separated by a /OPT statement.

## Parameters: Compile Step - VSCOBOL

```
/OPT  [parms...]   (see below for possible parameters)
```

The specified parameters should be separated by commas, and the last parameter must be followed by a

blank. A partial list of parameters and their MUSIC defaults is shown here. Additional parameters may be found in the *IBM COBOL Programmer's Guide*. The following parameters cannot be used: SYMDMP, RESIDENT, DYNAM, ENDJOB, TEST.

BUF= The amount of main storage allocated to buffers. The default value is BUF=16000, which will allow small and medium size source programs to be compiled relatively quickly. If the message INSUFFICIENT CORE FOR THIS JOB is printed, the user may reduce this value (for example, /OPT BUF=6000).

NOSOURCE Program source statements not to be listed. This is the default for jobs run at the workstation.

SOURCE Program source statements to be printed on SYSPRINT. This is the default for jobs run from batch.

NODECK No object module to be produced. This is the default.

DECK An object module is to be written on SYSPUNCH.

NOSEQ The compiler is not to do sequence checking on the source program statements. This is the default.

SEQ The compiler is to do sequence checking on the source program statements.

*Notes:*

1. Multiple /OPT statements may be used if desired. For other parameters which may be used on the /OPT statement, refer to the IBM COBOL Programmer's Guide publication.

2. /FILE statements with ddnames of COB.SYSIN, COB.SYSPRINT, and COB.SYSPUNCH can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions.

## Parameters: Loader Step

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,DUMP][,CDUMP][,DEBUG]
      [FULMAP]

          [,FILRFM][,NOSEARCH][,IOTRACE[(C)]][,SVCTRACE[(C)]]
```

Parameters can be separated by one or more commas or blanks.

MAP List a storage map of the loaded program.

FULMAP List an extended storage map of the loaded program.

NOGO Do not load or execute the program.

LET Allow the program to be executed even if unresolved external references (such as missing

subroutines) are found.

NOPRINT Informative and diagnostic messages are not to be produced.

DUMP Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump.

CDUMP Request a conversational trace and dump when an error occurs.

DEBUG Load the DEBUG program into memory for debugging.

FILRFM Supply the RECFM V if the file is V/VC.

NOSEARCH Do not search the subroutine library for unresolved routines.

IOTRACE Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed.

SVCTRACE Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed.

*Notes:*

1. Multiple /JOB statements may be used if desired.

2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

## Parameters: Go Step - VSCOBOL

Parameters can be passed to the GO step by specifying a "/PARM xxx" statement at the beginning of the job. *xxx* is the parameter desired.

SVC instructions and I/O operations can be traced during execution of the program. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement. The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the description of /JOB statement above.)

## Title Lines - VSCOBOL

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

# References - VSCOBOL

*IBM VS COBOL for OS/VS* (GC26-3857)

*OS/VS COBOL Programmer's Guide* (SC28-6483)

# Examples - VSCOBOL

1. An COBOL job with compiler and loader options. /FILE statements for GO step (execution time) ddnames are also defined.

```
/FILE GO.CARDIN RDR
/FILE GO.PRINTOUT PRT
/LOAD COBOL
/OPT DECK
/JOB MAP
      COBOL Source Program
/OPT
      COBOL Source Subprogram
/DATA
      Data
```

2. An COBOL program which reads data from SYSIN (data following /DATA), writes them on SYSPRINT (printed output), SYSPUNCH (holding file), and FILE01 (a temporary UDS file). At end-of-file on SYSIN, additional data is read conversationally from CONSOLE (workstation input) until the user signals end-of-file on the workstation by typing /EOF. Then the disk data set FILE01 is read back in and listed.

```
/FILE FILE01 UDS(&&TEMP) NREC(100)
/LOAD COBOL
       ID DIVISION.
       PROGRAM-ID. COBOL-IO-TEST.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CARD-FILE ASSIGN TO UR-S-SYSIN.
           SELECT DISK-FILE ASSIGN TO DA-S-FILE01.
           SELECT CON-FILE ASSIGN TO UT-S-CONSOLE.
           SELECT PRINT-FILE ASSIGN TO UR-S-SYSPRINT.
           SELECT PUNCH-FILE ASSIGN TO UR-S-SYSPUNCH.
       DATA DIVISION.
       FILE SECTION.
       FD  DISK-FILE
           LABEL RECORDS ARE STANDARD
           RECORDING MODE IS F.
       01  DISK-REC  PIC X(80).
       FD  CON-FILE
           LABEL RECORDS ARE OMITTED.
       01  CON-REC  PIC  X(80).
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  CARD-REC  PIC X(80).
       FD  PRINT-FILE
```

```
           LABEL RECORDS ARE OMITTED.
       01  PRINT-REC.
          05  CC    PIC  X.
          05  PRINT-LINE   PIC  X(132).
       FD  PUNCH-FILE
           LABEL RECORDS ARE OMITTED.
       01  PUNCH-REC  PIC  X(80).
       WORKING-STORAGE SECTION.
       77  HOLD1  PIC  9(4)  VALUE ZERO  COMP-3.
       PROCEDURE DIVISION.
           OPEN INPUT CARD-FILE OUTPUT PRINT-FILE, PUNCH-FILE
           DISK-FILE.
       START-HERE.
           READ CARD-FILE AT END GO TO EOF.
           MOVE CARD-REC TO PRINT-LINE.
           WRITE PRINT-REC AFTER POSITIONING 2 LINES.
           WRITE PUNCH-REC FROM CARD-REC.
           WRITE DISK-REC FROM CARD-REC.
           GO TO START-HERE.
       EOF.  DISPLAY ' THIS IS THE END NOW TRY INPUT FROM CONSOLE'.
           CLOSE CARD-FILE PRINT-FILE PUNCH-FILE.
             OPEN INPUT CON-FILE.
       CON-SECT.
           READ CON-FILE AT END GO TO VERY-END.
           DISPLAY 'FROM CONSOLE ' CON-REC.
           GO TO CON-SECT.
       VERY-END.
           CLOSE CON-FILE DISK-FILE. DISPLAY 'CONV END'.
           OPEN INPUT DISK-FILE.
           DISPLAY ' THE FOLLOWING RECORDS ARE FROM DISK FILE'.
       DISK-READ.
           READ DISK-FILE AT END GO TO FINAL-EXIT.
           DISPLAY ' RECORD FROM DISK ' DISK-REC.
           GO TO DISK-READ.
       FINAL-EXIT.
           CLOSE DISK-FILE. DISPLAY ' THATS ALL FOLKS..'.
           STOP RUN.
/DATA
FIRST DATA CARD
SECOND DATA CARD
THIRD DATA CARD
LAST DATA CARD
```

# COBOL (Loader) - COBLG

COBLG (ANS COBOL Load and Go) is the name used to access the loader. The /LOAD COBLG statement specifies that the lines following consist exclusively of object modules created by ANS COBOL and/or VS Assembler. (This statement is functionally identical to the /LOAD ASMLG statement.)

## Usage - COBOLG

This loader is invoked by the use of the /LOAD COBLG or /LOAD ASMLG statement. This statement can be followed by a /OPT SYSIN statement specifying the input unit number and by a /JOB statement specifying parameters for the loading step.

## Parameters

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,NOSEARCH][,FILRFM]
      [FULMAP]
```

Parameters can be separated by one or more commas or blanks.

| | |
|---|---|
| MAP | List a storage map of the loaded program. |
| FULMAP | List an extended storage map of the loaded program. |
| NOGO | Do not load or execute the program. |
| LET | Allow the program to be executed even if unresolved external references (such as missing subroutines) are found. |
| NOPRINT | Informative and diagnostic messages are not to be produced. |
| NOSEARCH | Do not search the subroutine library for unresolved routines. |
| FILRFM | Supplies a RECFM V if the file is V/VC. |

*Notes:*

1. Multiple /JOB statements may be used if desired.

2. The DEBUG, CDUMP, DUMP, IOTRACE and SVCTRACE parameters for OS-mode processors are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

```
/OPT  [SYSIN=n]
```

SYSIN=n Specifies that input is to be taken from the MUSIC unit number n. A /FILE statement defining MUSIC I/O unit n as the appropriate input file must appear at the beginning of the job. When an end-of-file or a /DATA statement is encountered on this input data set, further input is taken from the normal input stream.

# FORTRAN Compiler - VSFORT

Programs written in the Fortran language can be compiled and run using the IBM VS Fortran Compiler and Library. VS Fortran supports both the Fortran 77 and Fortran 66 language standards. Fortran 77, the newer standard, is the default. To request Fortran 66, you must specify the compiler option LANGLVL(66).

MUSIC has a tutorial for learning VS Fortran. Type "TUT" in *Go mode or select the "Tutorials" item for the FSI main menu.

## Available Versions - VSFORT

Each MUSIC/SP system provides either of two versions of VS Fortran (but not both): (1) VS Fortran Version 1 (Release 4) Compiler and Library, program number 5748-FO3, or (2) VS Fortran Version 2 Compiler, Library and Interactive Debug, program number 5668-806. Which version is present is an installation option and varies from site to site. Both versions are invoked by the /LOAD VSFORT control statement.

To see which version is available on your system, run the following job and examine the first digit of the level number in the heading line of the output:

```
/LOAD VSFORT
/JOB NOEDIT
      STOP
      END
```

Version 2 is compatible with Version 1, in that it can run all object modules and load modules produced by Version 1. Version 2 accepts Fortran source prepared for Version 1, but the reverse is not true.

VS Fortran Version 2 provides the following enhancements: 31-character variable names, language keywords in mixed upper and lower case (it is not case sensitive), an Intercompilation Analyzer (ICA), graphic relational operators (combinations of "<", ">" and "="), a source level Interactive Debug Facility (IAD), and other features not present in Version 1.

## Interactive Debug - VSFORT

With VS Fortran Version 2, MUSIC users can debug and test their programs at the source language level using either IBM's Interactive Debug (IAD) in line mode or MUSIC's full-screen VS Fortran Interactive Debug (TESTF). TESTF is described in the next section of this chapter. IAD is described later in this section. With VS Fortran Version 1, TESTF is available but IAD is not.

## Control Statements - VSFORT

VS Fortran in invoked by the /LOAD VSFORT statement. The VSFORT processor accepts source and/or object modules as input, compiles the source, invokes the OS-mode Loader to load the object modules and required library routines into memory, and executes the program. Both the compile and execution steps run in MVS (OS) simulation mode.

The /LOAD VSFORT statement can be preceded by a /SYS statement specifying job region size, by a /PARM statement specifying execution-time parameters (which may include the option FDEBUG to invoke Interactive Debug - IAD), and by /FILE statements defining files for the program. The /LOAD VSFORT statement can be followed by a /OPT statement specifying compiler options, and by a /JOB statement specifying Loader and MVS simulator options.

Control statements are used as follows:

```
/SYS REGION=n                <--- job region size in K
/PARM xxxxx                  <--- run-time parameters
/FILE nn NAME(filename) ...  <--- 0 or more /FILE statements
/FILE ddname NAME(filename) ...
/LOAD VSFORT
/JOB option,option,...       <--- Loader and MVS simulator options
/OPT option,option,...       <--- compiler options
...source program and/or object modules...
/DATA
...data records (Fortran unit 5)...
```

Fortran source, object modules, and data records can be specified by /INCLUDE statements pointing to other files, or the source or data can be contained in the same file as /LOAD VSFORT.

Most compiles require a region size of at least 512K (/SYS REGION=512). Large programs may require 1024K or more.

# Defining Program Files - VSFORT

A file is connected to a program by a unit number (also called a data set reference number), which is a number from 1 to 99, or by a ddname (data definition name), which is a 1 to 8-character name. The Fortran convention for ddnames is FTnnF001, where nn is a 2-digit number from 01 to 99. Fortran source statements such as READ and WRITE refer to unit numbers. A Fortran OPEN statement can be used to associate a unit number with a ddname. If no OPEN statement is used, unit number nn is automatically associated with ddname FTnnF001.

Unit numbers and ddnames are defined on /FILE statements which precede the /LOAD statement:

```
/FILE nn NAME(filename) options          <--- unit number

/FILE ddname NAME(filename) options      <--- ddname
```

On a /FILE, ddname FTnnF001 is equivalent to unit number nn. Think of unit number nn on a /FILE as shorthand for ddname FTnnF001.

The following unit numbers and ddnames are automatically defined for VS Fortran jobs:

| Ddname | Unit | Record Length | Definition |
|---|---|---|---|
| FT05F001 | 5 | 80 | Input data following /DATA |
| FT06F001 | 6 | 133 | Output to the workstation (for batch jobs: output to printer) |
| FT09F001 | 9 | 80 | Conversional input from the workstation (for batch jobs: same as unit 5) |

End of data can be indicated on conversational input from the workstation by typing /eof.

## Compiler Options on the /OPT Statement - VSFORT

Options for the VS Fortran compiler are specified on a /OPT statement. The options are separated by commas. If necessary, more than one /OPT can be used; their effect is cumulative.

The available options are described in the appropriate VS Fortran Programming Guide publication. MUSIC uses the standard IBM defaults. Some common compiler options are:

NOSOURCE
SOURCE — Produces a source listing on ddname SYSPRINT (the workstation by default). Default is NOSOURCE for workstation jobs and SOURCE for batch jobs.

NODECK
DECK — Produces an object module on ddname SYSPUNCH. The default is NODECK.

FLAG(I)
FLAG(W)
FLAG(E)
FLAG(S) — Specifies which level of compiler messages will be produced: all (I), warnings and above (W), errors and above (E), or only severe errors (S). The default is FLAG(I).

OPT(n) — Optimization level, 0 to 3. OPT(0) is no optimization, which gives faster compiles but produces less efficient code. OPT(1) is recommended for most production programs. The default is OPT(0).

NOSDUMP
SDUMP — Incorporates information on source variable names and statement numbers into the object module. This makes the program bigger but enables symbolic dumps and debugging (IAD) at execution time. The default is SDUMP.

CHARLEN(n) — Maximum length of CHARACTER variables. The default is CHARLEN(500).

LANGLVL(66)
LANGLVL(77) — The Fortran language level to be accepted. The default is LANGLVL(77). If necessary, specify LANGLVL(66) for compatibility with Fortran G1 source.

NOMAP
MAP — Produces a table of variable names and statement labels. The default is NOMAP.

NOXREF
XREF — Produces a table of variable names and statement labels, plus the statement numbers where each is referenced. The default is NOXREF.

NOLIST
LIST — Shows the assembler instructions generated by the compiler. This is useful if you wish to use the assembler-level Debug Facility to debug your program. The default is NOLIST.

## Loader and MVS Simulator Options on the /JOB Statement

Options for the MUSIC Loader and for the execution-time MVS simulator are specified on a /JOB statement. The options are separated by commas or blanks. If necessary, more than one /JOB can be used; their effect is cumulative. When using /LOAD XMON to execute a load module, the same options may be used on the program name statement, after the load module ddname or N(filename) specification.

The available /JOB options are:

| | |
|---|---|
| MAP | Produces a Loader storage map. |
| FULMAP | Produces an extended Loader storage map. |
| NOGO | Compile only. Do no load or execute the program. |
| LET | Allows the program to execute even if unresolved external references (such as missing subroutines) are found during the Loader step. |
| NOSEARCH | Tells the Loader not to search the Subroutine Library for unresolved external references occurring in the program. With this option, the Loader does not load any subroutines from the Subroutine Library. |
| NOPRINT | Omit Loader information messages. |
| SVCTRACE | Traces SVC (supervisor call) instructions during program execution. Abbreviation: SVCTR |
| IOTRACE | Traces input/output operations during program execution. Abbreviation: IOTR |
| DUMP | Produces a storage dump if the program ends abnormally or is terminated by an error detected by the MVS simulator. Abbreviation: DU |
| CDUMP | Allow interactive display of storage if the program ends abnormally or is terminated by an error detected by the MVS simulator. If used in combination with SVCTRACE or IOTRACE, CDUMP stops the program after each SVC or I/O and allows you to display and modify storage interactively. Abbreviation: CDU. |
| DEBUG | Invokes the MUSIC assembler-level Debug Facility at the start of program execution. |
| NONRENT | Forces re-entrant modules in the Link Pack Area (LPA) to be loaded into the user region. This is required if you wish to use the assembler-level Debug Facility to trace such modules or set break points in them. |
| NOEDIT | Suppresses editing of compiler output to the workstation. Normally MUSIC omits or shortens some headings and other information messages during compile. The NOEDIT option allows the full output to appear. |
| V(n) | This option, where n is a record length value, causes the VS Fortran Library routines to see variable-length MUSIC files (record format V or VC) as fixed-length MVS files (record format F or FB), with a record length equal to the larger of n and f, where f is the MUSIC record length of the file. This option applies only to ddnames FTnnF001. For example, /JOB V(300) causes VS Fortran to treat a RECFM(VC) file with MUSIC record length 200 as a fixed format file with record length 300. This allows new output records to be up to 300 bytes long and avoids the 8 bytes of control information (MVS record and block descriptor words) at the start of each record. The option does not affect the actual record format of the resulting MUSIC file. Note that a new V or VC file has a MUSIC record length of 0. V(256) is automatically assumed for FT06F001. |
| FILRFM | This option causes MUSIC record format V or VC files to be treated as variable length (MVS record format VB) files by the VS Fortran Library routines. This option may be required in order to produce the MVS record and block descriptor words at the start of each logical record. |

## Execution-Time Parameters - VSFORT

Fortran run-time options, if required, are specified on a /PARM statement preceding the /LOAD VSFORT or /LOAD XMON statement. All run-time options listed in the VS Fortran Programming Guide can be specified, except that the DEBUG option must be entered as FDEBUG on MUSIC. Thus, the statement for invoking Interactive Debug (IAD) at run time is /PARM FDEBUG (not /PARM DEBUG).

Any options or parameters on /PARM are also accessible to the program via a call to the system subroutines PARM or PARMLC. PARMLC is the same as PARM, except that the parameter string is not converted to upper case.

## Block Letter Titles - VSFORT

For a compile in a batch job, one or two lines of block letter titles can be printed preceding the source listing. Each title line is from 1 to 10 characters. The titles are printed twice on consecutive pages, so that one will always be face up after page tear off. The titles are ignored if the job is run at a workstation. The titles are specified by control statements following the /LOAD statement:

```
=TITLE xxxxxxxxxx             <--- first block letter title

=TITLE2 yyyyyyyyyy            <--- second block letter title
```

For example:

```
/LOAD VSFORT
/OPT SOURCE
=TITLE PROG5 VER1
/INCLUDE PROG5.S         <--- program source file
```

## Using Object Modules and Load Modules - VSFORT

To create an object module in file prog.obj, use the compiler DECK option as in this example:

```
/SYS REGION=1024
/FILE SYSPUNCH NAME(prog.obj) NEW    <--- object output
/LOAD VSFORT
/JOB NOGO
/OPT DECK,OPT(1)        <--- other options as desired
/INCLUDE prog.s         <--- program source file
```

The MUSIC convention is that source module file names end in ".S", object module file names end in ".OBJ", and load module file names end in ".LMOD". You may use a different convention if you wish.

Once you have object modules for all the routines making up your program, you can create a load module in file prog.lmod as in this example:

```
/FILE LMOD NAME(prog.lmod) NEW SPACE(200)
/LOAD LKED
/JOB MAP,NOGO,MODE=OS,NAME=progname
.ORG 4A00
/INCLUDE prog.obj
```

```
/INCLUDE sub1.obj
/INCLUDE sub2.OBJ
```

To execute the program using the load module, use the following control statements:

```
/SYS REGION=n                <--- desired region size
/FILE ...                    <--- /FILE statements as needed
/LOAD XMON
progname N(prog.lmod) option,option,...  <--- same opts as /JOB
...data records if any...
```

Using a load module to execute your program gives much faster job start up.

# Usage Notes - VSFORT

1. Since VS Fortran programs can invoke features of the MVS operating system which are not supported by MUSIC's MVS simulator, it is possible to compile Fortran programs which do not execute successfully on MUSIC.

2. MUSIC supports use of VSAM files with VS Fortran. Refer to the section on VSAM elsewhere in this manual.

3. Features of VS Fortran that depend on MVS dynamic file allocation are not supported on MUSIC. However, the MUSIC subroutines OPNFIL and CLSFIL can be used to access MUSIC files dynamically (without using a /FILE statement). Refer to the chapter on System Subroutines in this manual.

4. Files for VS Fortran direct access must be created as record format F (fixed length, uncompressed). Each direct access record starts on a new 512-byte block on disk, and occupies one or more blocks. This storage method is inefficient in terms of disk space when the record length is small (256 bytes or less). The feature of VS Fortran that automatically formats a new direct access file is not available on MUSIC. The OPEN statement must not specify STATUS='NEW'; omit the STATUS keyword or specify STATUS='OLD'. A new direct access file must be initialized using the ZERO.FILE utility (or equivalent) before being used in a VS Fortran program, unless the program writes all the records in order (1,2,3,...) to the end of the file. Once the file has been initialized, it can be accessed randomly. This example shows creating and initializing a direct access file:

   ```
   /FILE 1 NAME(DA.SAMP) NEW RECFM(F) LRECL(400) SPACE(100) NORLSE
   /INC ZERO.FILE
   ```

5. The VS Fortran INCLUDE statement can be used to incorporate source from an external file into the program. The following format of INCLUDE must be used:

   ```
   INCLUDE (member) n
   ```

   "member" is a 1 to 8-character member name. "n" is an optional value which the compiler uses to decide whether or not to include the source. By default, the contents of MUSIC file member.VSFORT is included into the program. To use a different naming convention for included files, provide a /FILE SYSLIB containing a PDS parameter. The default SYSLIB is /FILE SYSLIB PDS(*.VSFORT). INCLUDE statements must not be nested; that is, the included file may not contain another INCLUDE statement. The MUSIC /INCLUDE statement (which allows nesting) can be used as an alternative to the VS Fortran INCLUDE statement.

# Interlanguage Communication - VSFORT

It is sometimes desirable to invoke subprograms written in other programming languages. A VS FORTRAN main program may call subroutines written in Assembler, PL/I, VS/PASCAL, COBOL, and VS FORTRAN.

You may also invoke VS FORTRAN subroutines from Assembler, COBOL, PL/I, and VS/PASCAL. When calling VS FORTRAN subroutines from other languages (where the main program is NOT VS FORTRAN), you must call a subroutine VSFINT before any calls to VS FORTRAN routines. This routine (VSFINT) performs library initialization for the VS FORTRAN environment.

The format of the VSFINT call is:

**VS/PASCAL**

```
 var
   I            :  INTEGER;
  PROCEDURE VSFINT(VAR I : INTEGER);
   FORTRAN;
begin
   VSFINT(I);
     .
     .
```

**VS/COBOL**

```
ID DIVISION.
PROGRAM-ID. COBOL-TEST.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77     AREAL  USAGE IS COMPUTATIONAL-2.
PROCEDURE DIVISION.
START-HERE.
    CALL 'VSFINT'  USING AREAL.
         .
         .
```

**PL/I**

```
TOSQ: PROC OPTIONS(MAIN);
      DCL SQUARE ENTRY EXTERNAL;
      DCL VSFINT ENTRY EXTERNAL;
      IREAL=4;
      CALL VSFINT(IREAL);
           .
           .
```

There is an argument passed in the call to VSFINT. This argument is NOT used by the subroutine, and can be of any value.

# MUSIC Extensions to NAMELIST Input

MUSIC allows a more convenient form of Fortran NAMELIST input data if the object module MUSNL.OBJ is included with the program. NAMELIST output is not affected. If MUSNL.OBJ is not present, the standard form of NAMELIST input, as described in the language manual, is used.

When /INCLUDE MUSNL.OBJ is placed after the Fortran program, or in the Linkage Editor input when creating a load module, the NAMELIST start (&name) and end (&END) indicators are not used in input data records. The first data item can start at or after column 1 and can extend to column 80. To continue a record, end it with a comma. For a logical variable, for example SWITCH, specifying "SWITCH" is equivalent to "SWITCH=.TRUE." and "NOSWITCH" is equivalent to "SWITCH=.FALSE." When an character string is specified for an array, the remaining elements of the array are set to blanks.

Subscripted array items, floating point exponents, REAL*16 and complex values may not be used. A method of assigning values to particular items of an array is, for example, A=10,3*,20, which sets A(1)=10 and A(5)=20 but does not change A(2) through A(4). The ERRSET routine cannot be used to get control of conversion or other errors. For VS Fortran Version 2, variable names may be up to 31 characters long.

The sample program given below shows an example of MUSIC NAMELIST input.

# VS Fortran Interactive Debug (IAD)

The IBM version of Interactive Debug (IAD) is invoked by specifying the run-time option FDEBUG. This is done by placing the statement /PARM FDEBUG before the /LOAD VSFORT or /LOAD XMON statement.

IAD is provided only with VS Fortran Version 2. Since ISPF is not supported on MUSIC, only the line and batch modes of IAD are available. Full screen mode is not available. If full screen mode debugging is needed or you wish to debug with VS Fortran Version 1, use the MUSIC version of VS Fortran Interactive Debug (TESTF).

The only requirement for using IAD is that the routines to be debugged must be compiled with the SDUMP option (which is the default). Also, the compiler option OPT(0) is recommended, to suppress optimization. Some extra ddnames are required for IAD, and the job region size must be increased by at least 500K. To simplify the control statements, file FDEBUG is provided, which contains /PARM FDEBUG, specifies a region size of 1024K, and defines the AFF-- ddnames required for IAD. In most cases, it is sufficient to place the statement /INCLUDE FDEBUG before the /LOAD.

When IAD is run on batch, the AFFIN file must not be empty. Do this by providing an overriding /FILE AFFIN pointing to the file containing the IAD commands.

The IAD HELP command invokes the MUSIC Help Facility to display full screen help information. The IAD SYSCMD command (abbreviation SYS) can be used to enter a MUSIC command while debugging. For example, "sys edit prog.listing". SYSCMD cannot be used within an IAD command list. To signal an attention interrupt to IAD, press the BREAK key (PA1 on a 3270-type workstation) and enter a blank line while in MUSIC attention mode. For the IAD TERMIO command to take effect, the desired unit numbers must be specified by the DEBUNIT run-time option. For example:

```
/PARM FDEBUG,DEBUNIT(5,6,9)
```

The following is a sample Interactive Debug session.

```
/list fdebug.sample
*In progress
```

```
/INCLUDE FDEBUG
/LOAD VSFORT
      READ(5,*) DIAM
      CALL CALC(DIAM,CIRCUM,AREA)
      WRITE(6,10) DIAM,CIRCUM,AREA
10    FORMAT(' DIAMETER=',F8.3,'  CIRCUMFERENCE=',F10.4,
     * '  AREA=',F10.4)
      STOP
      END
      SUBROUTINE CALC(X,C,A)
      DATA PI/3.14159/
      C=X*PI
      A=(X/2)**2*PI
      RETURN
      END
/DATA
3.51
*End
*Go

fdebug.sample
*In progress

**MAIN** END OF COMPILATION 1 ******

**CALC** END OF COMPILATION 2 ******
004F50 BYTES USED
EXECUTION BEGINS
VS FORTRAN VERSION 2 RELEASE 4 INTERACTIVE DEBUG
5668-806 (C) COPYRIGHT IBM CORP. 1985, 1989
LICENSED MATERIALS-PROPERTY OF IBM
WHERE: MAIN.1
FORTIAD
?
listsubs
PROGRAM UNIT                     COMPILER   OPT   HOOKED TIMING
MAIN                            VSF 2.4.0    0      YES    OFF
CALC                            VSF 2.4.0    0      YES    OFF
FORTIAD
?
at calc.4
FORTIAD
?
go
AT: CALC.4
FORTIAD
?
list (calc.x,calc.c,calc.a)
CALC.X           =  3.51000023
CALC.C           =  11.0269814
CALC.A           = 0.000000000E+00
FORTIAD
?
go
DIAMETER=   3.510  CIRCUMFERENCE=   11.0270  AREA=    9.6762
PROGRAM HAS TERMINATED; RC ( 0)
```

```
FORTIAD
?
listfreq
STATEMENT                                          FREQUENCY
MAIN.ENTRY                                          NO HOOK
MAIN.EXIT                                           NO HOOK
MAIN.1                                              1
MAIN.2                                              1
MAIN.3                                              1
MAIN.5                                              1
MAIN.6                                              0
FORTIAD
?
quit
*End
*Go
```

## Separation Tool - VSFORT

The VS Fortran Separation Tool utility program is used to create the re-entrant and non-reentrant parts of a VS Fortran program compiled with the RENT option. Refer to the VS Fortran Programming Guide. The re-entrant part of a program could be put into MUSIC's Link Pack Area to provide better performance for high use programs.

Control statements for running the Separation Tool are:

```
/PARM name              <--- for "assigned name" form only
/FILE SYSUT1 NAME(nonrent.obj) NEW
/FILE SYSUT2 NAME(rent.obj) NEW
/INCLUDE VSFORT.SEPTOOL
/INCLUDE prog.obj       <--- object from compile with RENT
/INCLUDE sub1.obj
...
```

## Alternate Math Library (Version 1 Only)

To use the Alternate Math Subroutine Library supplied with VS Fortran Version 1, insert the following statement after your Fortran source (or after your object modules when creating a load module):

```
/INCLUDE MATH.ALTLIB
```

## Sample Program - VSFORT

```
list vsfort.sample
*In progress
/SYS REGION=512
/LOAD VSFORT
C  VS FORTRAN SAMPLE PROGRAM.
C  CALCULATES RADIUS, CIRCUMFERENCE, AND AREA OF A CIRCLE,
C  GIVEN THE DIAMETER.
```

```
C  ANSWERS CAN BE REQUESTED IN HIGH OR LOW PRECISION.
C  INPUT IS BY MUSIC NAMELIST.
      REAL*8 DIAM,RADIUS,CIRCUM,AREA
      LOGICAL LOW,HIGH
      NAMELIST /DATAIN/ DIAM,LOW,HIGH
100   WRITE(6,*) 'Enter: DIAM=value,LOW  OR  DIAM=value,HIGH'
      LOW=.FALSE.
      HIGH=.FALSE.
      READ(9,DATAIN,END=200)
      IF(.NOT.(LOW.OR.HIGH)) LOW=.TRUE.
      CALL CALC(DIAM,RADIUS,CIRCUM,AREA)
      IF(LOW) WRITE(6,10) DIAM,RADIUS,CIRCUM,AREA
10    FORMAT(' DIAMETER=     ',F9.4,3X,'RADIUS=    ',F9.4/
     *       ' CIRCUMFERENCE=',F9.4,3X,'AREA=      ',F9.4)
      IF(HIGH) WRITE(6,20) DIAM,RADIUS,CIRCUM,AREA
20    FORMAT(' DIAMETER=     ',F12.6,3X,'RADIUS=    ',F12.6/
     *       ' CIRCUMFERENCE=',F12.6,3X,'AREA=      ',F12.6)
      GO TO 100
200   STOP
      END
      SUBROUTINE CALC(X,R,C,A)
      REAL*8 X,R,C,A,PI/3.14159265/
      R=X/2
      C=X*PI
      A=R**2*PI
      RETURN
      END
/INCLUDE MUSNL.OBJ
*End
*Go

vsfort.sample
*In progress

**MAIN** END OF COMPILATION 1 ******

**CALC** END OF COMPILATION 2 ******
006010 BYTES USED
EXECUTION BEGINS
Enter: DIAM=value,LOW  OR  DIAM=value,HIGH
?
diam=3.51,low
DIAMETER=        3.5100   RADIUS=       1.7550
CIRCUMFERENCE=  11.0270   AREA=         9.6762
Enter: DIAM=value,LOW  OR  DIAM=value,HIGH
?
diam=3.51,high
DIAMETER=         3.510000   RADIUS=        1.755000
CIRCUMFERENCE=   11.026991   AREA=          9.676185
Enter: DIAM=value,LOW  OR  DIAM=value,HIGH
?
/eof
*End
*Go
```

# References - VSFORT

**Manuals for VS Fortran Version 1**

*VS Fortran Language and Library Reference*, SC26-4119.

*VS Fortran Programming Guide*, SC26-4118.

**Manuals for VS Fortran Version 2**

*VS Fortran Version 2 Language and Library Reference*, SC26-4221.

*VS Fortran Version 2 Programming Guide*, SC26-4222.

*VS Fortran Version 2 Interactive Debug Guide and Reference*, SC26-4223.

*VS Fortran Version 2 Master Index and Glossary*, SC26-4603.

*VS Fortran Version 2 Reference Summary*, SX26-3751 (booklet).

# MUSIC/SP VS/FORTRAN Debugger - TESTF

TESTF, the VS/FORTRAN Full Screen Debugger, enables you to control your VS/FORTRAN program completely during the test and debugging stages of program development. You can set break-points, inspect and alter variable values, scan the source for text, generate a trace listing (which includes actual source statements), re-start execution at another instruction (within the current module), cause execution to cease at entry and/or exit of routines, and manipulate arrays. As well, TESTF is integrated with DEBUG, the machine level debugger. This enables you to view and execute the actual machine instructions that implement your VS/FORTRAN instructions. Of course, MUSIC commands can be issued from TESTF.

## Invoking TESTF (VS Fortran Debugger)

The easiest way of invoking TESTF is to use the TESTF REXX exec. A simple example: in order to test your VS/FORTRAN program, called ASSIGN1, enter the following:

```
testf  assign1
```

```
---- VS/FORTRAN Interactive Debug ---- Where: MAIN     @ ISN:       6  Line:    7
Command ==>
Data Window Displayed
--ISN--*....*...1.........2.........3.........4.........5.........6.........7..
    2        INTEGER*2    I2(2,2,2,2)
    3        REAL         X(4,4)
    4        INTEGER I1,II1,III1,J,K,L
    5        LOGICAL  A,B,C
       C
    6  10    A = .TRUE.
    7        B = (.NOT.A)
    8        C = .FALSE.
       C
    9        I1 = 10
   10        II1 = 100
Module---Variable-Address-Type-----Len-Atr--Hex Value--Value by Type ----------
MAIN     II1      004990  INTEGER4  4       00000000    0
MAIN     III1     004994  INTEGER4  4       00000000    0
MAIN     CHRVAR   005800  CHAR     50    A  E3C8C9E240  THIS IS THE  (1,1) ELEME

------------------------------------------------------------------- ----------
1:Help   2:Run Pgm        5:Single Step    7:Page Up    4:Arrays    10:Debug
3:End    11:Animate Pgm   9:Break Point    8:Page Down  6:Data      12:Retrieve
```

*Figure 8.6 - Example of MUSIC's VS/FORTRAN Debugger Display*

The following is a more detailed description of the TESTF exec.
The complete format for invoking TESTF is as follows:

```
TESTF source OBJECT(...) PARMS(...) REGION(...) FILES(...) LINK
             LISTING(name) DATA(name)
```

where the above parameters have the following interpretation:

| | |
|---|---|
| source-name | name of source file to compile and test; if omitted then the exec assumes that you wish to run from object modules (using the OBJECT parameter) |
| OBJECT(...) | name of object file to be included at run or link edit time; can contain /INCLUDE statements |
| LINK | requests that a load module be generated and that it subsequently be executed |
| FILES(...) | name of a model file containing /FILE statements that will be used when the program is executed. |
| LISTING(...) | name of a listing file if no source file was provided. The file specified will be used for the listing if a source file was specified. |
| REGION(...) | specifies the region size to be used for all jobs |
| PARMS(...) | application parameters for your VS/FORTRAN program |
| DATA(name) | specifies the name of a data file that will be accessible via READs on unit 5. Note that this file should already exist; otherwise, an error will result. |

If you are not using the TESTF exec, then you should be aware of the following. Modules to be debugged MUST be compiled with the TEST and SDUMP compiler options. You must request a source listing, to be saved in a file with LRECL 133. In order to use the Debugger, set up JCL as in one of the following 3 examples (compile and go, load-and-go, and exec from load module).

Note that the VS/FORTRAN Debugger first looks for a DDname DBGLIST for the source listing, and then (if not found) searches for SYSPRINT. So if there is a conflict with using SYSPRINT, then utilize DBGLIST instead.

Also, note that in the case of a large application, not all of the application need be debugged. You can elect to debug only certain modules; the Debugger will "wake up" within these modules. However, you must ensure that the VS/FORTRAN environment has been established; this requires that you have a MAIN program or at least call the VS/FORTRAN initialization routines. Refer to the VS/FORTRAN Programmer's Guide for more information on how to call VS/FORTRAN initialization routines, if you do not have a VS/FORTRAN main program.

Compile-and-go

```
/SYS REG=1500
/FILE SUBLIB  PDS(*DBG,*OS,*MUS)
/FILE SYSPRINT N(module.LST) LR(133) NEW(REPL)
/LOAD VSFORT
/OPT SOURCE,SDUMP,TEST
/INCLUDE module.S
```

Load-and-Go

```
/SYS REG=1500
/FILE SUBLIB  PDS(*DBG,*OS,*MUS)
/FILE SYSPRINT NAME(module.LST) SHR
/LOAD ASMLG
/INCLUDE module.S
```

Load Module Exec

```
1) Linking

   /FILE LMOD NAME(module.LMOD) NEW(REPL) LR(128) RECFM(F)
   /FILE SUBLIBOS  PDS(*DBG,*OS,*MUS)
   /LOAD LKED
   /JOB NOGO,MODE=OS,NAME=module
   /INCLUDE module.obj
2) Execution

   /SYS REG=1500
   /FILE SYSPRINT NAME(module.LST)  SHR
   /FILE LMOD NAME(module.LMOD)  SHR
   /LOAD XMON
   module
```

# VS Fortran Debugger Function Keys

F1 Help — Accesses the help text.

F2 Run — This key/command causes your program to resume executing, without returning control to the Debugger unless a breakpoint is encountered, an error occurs, or your program ends. Any breakpoints that have been set will be honoured, as will Entry and Exit stops (set via the Halt command). When the program terminates a message to this effect is placed in the message area.

If you have enabled tracing with the TRACE command, then a trace listing will be produced as execution proceeds.

F3 Exit — This key/command causes an immediate exit from both your program and the Debugger.

F4 Arrays — Pressing this key presents the Array display. The Array display allows you to display and modify values of an array. Placing the cursor on an array in the data display causes that array to be displayed.

In the Array display, you can modify the index values that appear on the same line as the array name, in order to display that array element as the first line. As you scroll up and down through the array, the index values are altered in order to display the index value for the top-most array element in the Array display. (Keep in mind that first index value changes the fastest, and so on; index values are in the same order as would be used in your program.)

F5 Single Step — Control is advanced to the next debuggable statement. Normally, this means that a single statement is executed, but when a routine is called that has not been compiled with the TEST option, the Debugger cannot step into the routine; thus the whole routine is executed. Any variables that have changed are updated on the screen.

If you place the cursor on a line and press F5 (STEP), then a temporary breakpoint is placed on that line, instead of the next instruction, and execution resumes, until that temporary breakpoint is encountered. In other words, your program will be executed up until that line.

A typical use of this feature is to step past a DO loop without having to set a breakpoint, then issuing the RUN command, and then resetting the breakpoint. Note that when using this feature, there is no guarantee that control will actually be returned to you at that statement. If the logic of your program is such that control never arrives at that statement, then the use of this feature is equivalent to a RUN command. If you have no other breakpoints, then your program will run to termination.

F6 Data

Pressing this key presents the Data display. The Data display allows you to display and modify values of variables known to the Debugger. Use the FIND VARIABLE (FV) command in order to locate a variable, and use the SHOW and HIDE commands in order to control which variables are displayed. Use options in the SET command as well in order to control how and when the variable list is modified.

In the Data display, you can modify a variable in both hexadecimal and "natural" type. In other words, if the variable is of character type, you can either type in hexadecimal codes for the characters, or the characters themselves.

F7 Up
F8 Down

Causes the either the Source Window or the data window to scroll up or down by one page. The Data Window is either the Data, Array, or Watch display. The Source Window contains the source lines being debugged. The window that is scrolled depends upon where the cursor is located. If the cursor is in the Source Window, then the Source Window is scrolled (note: the command line is considered part of the source window). If the cursor is in the Data Window, then the Data Window is scrolled.

If the cursor is in the Source Window on a source line, pressing F7 or F8 causes that line to be positioned at the bottom or top of the Source Window Display, respectively. The cursor is placed into the command area. This allows you to "fine tune" your position in the source listing, without having to use the LINE command.

F9 Set/Reset Break Points

Placing the cursor on a source line with an ISN (Internal Sequence Number) sets a break point at that line, indicated by the characters 'ISN' being replaced by 'Brk'. Repeating this caused the break point to be reset.

F10 Debug

This places you into Debug (machine level), all set up to Debug the machine language instructions for the current VS/FORTRAN instruction. To return to the VS/FORTRAN Debugger, press F2 (RUN) or F3.

The machine level Debug is integrated with the VS/FORTRAN Debugger, in that it understands the VS/FORTRAN TEST environment. The machine level debugger displays the current VS/FORTRAN instruction being executed in the message area. Pressing F3 returns to the VS/FORTRAN Debugger without executing the instruction, whereas F2 executes the machine instructions for that statement, and then returns to the VS/FORTRAN debugger.

Be careful not to alter 'Set System Mask (SSM)' instructions, as they provide ISN information to the Debugger. When you reach a BAL instruction in front of one of these, press F5 (Step). This will position you at the next instruction to be executed (The corresponding VS/FORTRAN statement will be updated, as well).

F11 Animated Execution

This key causes the Debugger to go into animated execution mode. Approximately once per second (default value - "Set Delay" can alter this), an instruction is executed and the screen is updated. The effect is that of continuously single-stepping

your program. In order to exit this mode, press Enter and animation will cease.

F12 Retrieve — This key retrieves previously entered commands into the command field. This allows you to re-issue commands after (possibly) correcting them, for example.

# VS Fortran Debugger Commands

Although only the full form of the command name is given here, all commands can be specified with the minimum number of characters necessary to unambiguously identify the command.

Some commands refer to the 'current module'. This means the module of your program that is currently executing. You can determine what this is by looking on the first line of the screen; the name after the label WHERE is the name of the 'current module'.

## Find

The Find command has a number of variants. Although the actual syntax is:

```
find search-type  arg
```

where 'search-type' is the function being requested, you can specify each variant of the FIND command by 2 characters, given below.

Find String (FS) — This command locates a string in the source listing, starting from the current line. If the string is not found, then the search is re-started from the top. Thus the effect is that the string is found, no matter where it is in the source; the search "wraps around", so to speak.

Syntax:

```
find string text          or     fs text
```

where 'text' is the string to be located. the current line.

Find Variable (FV) — The FIND VARIABLE command locates a variable in the variable list, starting from the first variable displayed. As with the string search, the search "wraps around" if the variable is not found between the current line and the bottom of the list. If a variable is hidden, it is made visible if it is found.

Note that by default, the search is restricted only to the current module. In order to search the entire variable list, use the '*' option for the module name.

Syntax:

```
find var name             or     fv name
find var * name           or     fv * name
find var mod name         or     fv mod name
```

where 'name' is the name of the variable to be located, 'mod' is the module name to which the search should be restricted, and '*' means locate a variable with the specified name, regardless of which module it is in.

Find Module (FM) — This command searches for the named module. In addition to subroutines which

have explicit names, the name MAIN is recognized as the name of the main program, even though it does not appear in the source. This is because the command operates on the results of an analysis of the source input. By the same token, any module not in the source listing cannot be found.

Syntax:

```
find module mod          or     fm mod
```

where 'mod' is the name of the module to be located.

Find Breakpoint (FB)

This command finds the next breakpoint in the source listing. If a breakpoint is not found between the current line and the end, then the search "wraps around" from the start of the source listing.

Syntax:

```
find break               or     fb
```

Find Label (FL)

This command locates the statement label specified. Unless modified by a module name, the search is restricted to the current module.

Syntax:

```
find label  #            or     fl #
find label  *  #         or     fl *   #
find label mod #         or     fl mod #
```

where '#' is the statement label, 'mod' is the name of a module, and '*' means all modules.

Find ISN (FI)

This command locates the Internal Sequence Number (ISN) specified. (An ISN is the number assigned to all FORTRAN statements, except comments, and appears to the left of the FORTRAN statement in the listing.) Unless modified by a module name, the search is restricted to the current module.

Syntax:

```
find isn  #            or     fi #
find isn  *  #         or     fi *   #
find isn mod #         or     fi mod #
```

where '#' is the ISN, 'mod' is the name of a module, and '*' means all modules.

## Line

This command allows you to specify exactly which line to display. Although the line numbers are NOT displayed on the screen, the line number of the top line of the current display is located in the upper right-hand corner of the screen. This command is intended as a convenience in moving quickly to a particular area of the source listing.

## Top/Bottom

'TOP' moves the source window to the top of the source listing. 'BOTTOM' moves the source window to the end of the source listing.

## Trace

This command controls whether tracing information is produced when your application program is running. Initially, tracing information is not produced. When TRACE is entered, it causes trace messages to be produced when you issue a RUN command or a STEP command (over more than 1 statement); this mode has been "enabled". (You can save these messages by issuing the MUSIC command /REC NEW before running TESTF, and issuing /REC OFF after you exit from TESTF. The trace messages will be in a file called @REC.000) Entering TRACE again causes this to be cancelled, or "disabled". The messages produced by this command tell you whether TRACE mode has been "enabled" (activated) or "disabled" (deactivated).

*Note:* Enabling TRACE, then entering the machine Debugger and tracing instructions there, causes the FORTRAN and machine trace outputs to be merged.

## Halt

This command allows you to run without breakpoints and to interrupt the execution of your program at the entry or exit from subroutines.

| | |
|---|---|
| no parameter | Entering HALT without a parameter displays the status of the HALT conditions, both ENTRY and EXIT (see below). |
| ENTRY | 'HALT ENTRY' causes the Debugger to interrupt your program whenever control enters a subroutine. In other words, you receive control in the Debugger at the beginning of every subroutine. |
| EXIT | 'HALT EXIT' causes the Debugger to interrupt your program whenever control exits a subroutine. In other words, you receive control in the Debugger at the end of every subroutine. |
| OFF | 'HALT OFF' resets both Halt conditions to inactive. |

Syntax:

```
halt
halt entry
halt exit
halt entry exit
halt off
```

## SET

This command allows you to tailor how the Debugger operates. Typically, these options control default actions of the Debugger.

```
Set  AutoShow
     AutoHide
     Bell
     Count
```

```
Delay
```

AutoShow 'SET AUTOSHOW' allows you to alternate between two modes of presenting variables.

The default mode is that only variables in the current module are displayed. When you call another module, the list of variables is cleared and only the variables in that module are displayed. This guarantees that the value in the variable 'I', for example, that you see refers to the variable I in the module currently being executed.

The other operating mode does NOT reset the variable list every time a subroutine is called. This allows you by default to retain all variables in the list of variables. Normally, this is not necessary or desirable.

AutoHide This option affects how the SHOW command (see below) operates. By default, when a SHOW command is issued, the list of variables is first cleared. 'SET AUTOHIDE' alternates between this mode of operation and the other mode whereby the list of variables is not cleared by default (you can do this via the HIDE *.* command, see below).

Thus to view only 2 or 3 variables constantly, issue the 'SET AUTOHIDE' command. Then issue 'HIDE *.*', and finally, issue your SHOW command (for example, SHOW I J K). This will fix I, J, and K in the variable window. If you wish these to be preserved across subroutine calls, then issue the 'SET AUTOSHOW' command as well.

Bell This option switches between never sounding the alarm and sounding the alarm when an error or unusual condition occurs.

Count This alters the maximum statement count (default 5000). When you run your program, the debugger counts the number of instructions executed. When this number is exceeded, the debugger interrupts your program and places a message on the display.

This feature allows you to run your program without fear that you will "lose control" and be caught in an loop. Likewise, if your program is looping, this feature allows you to discover where it is doing this.

The value specified with COUNT should be a positive number. To disable counting, specify a large positive number (i.e. 999999).

Delay This option alters the default delay time of 1 second between statements when using Animated Execution (see ANIMATE command). Specify the number of seconds to pause between VS/FORTRAN instructions; it must be a whole number bigger than 0.

## SHOW

This command allows you to control the variables displayed in the Variable window. (See also SET AUTOSHOW and SET AUTOHIDE.) You can specify a list of individual variables and/or templates. The '*' used as a variable or module name means all variables or all modules. If used as part of a name (for example, I*) it means all variables or modules starting with 'I'.

Entering SHOW without parameters causes the Debugger to report the number of variables shown, and the total number of variables known.

Syntax:

```
       +-- var-name         displays variable in current module
       |   module.var-name  displays variable in indicated module
  Show |   *.var-name       displays indicated variable in all modules
       |   module.*         displays all variables in indicated module
       +-- *.*              displays all variables, all modules
```

## HIDE

HIDE accepts the same format of parameters as does SHOW. However, variables that match the template are hidden from view, and not displayed.

Entering HIDE without parameters causes the Debugger to report the number of variables hidden, and the total number of variables known.

## GO

This command allows you to alter the next instruction to be executed. Normally, FORTRAN programs execute instructions sequentially, unless the flow of control is modified by a loop, GO TO, or IF-THEN-ELSE construction. Within the Debugger, however, for testing purposes it is sometimes useful to re-do a number of statements (perhaps when erroneous input is recognized and corrected). The GO command allows you to execute any instruction within the CURRENTLY EXECUTING module ONLY. Thus you cannot suddenly execute a statement in some subroutine that is not active.

Note: GO does not cause execution to commence. Use the RUN command for this. This gives you a chance to verify that the statement to be executed next is in fact the one you wanted to be executed.

Syntax:

```
go #
```

where '#' is the ISN (Internal Sequence Number, given to the left of the statement in the source listing).

## External Command

In order to execute a MUSIC command (say to Edit or View a file), preface the command in the command line with a leading '/'. Thus, to Edit your file called 'ASSIGN1', enter the following in the command line:

```
/edit assign1
```

When you have finished your Edit session, you will be returned to the Debugger.

# General Purpose Simulation System - GPSS

General Purpose Simulation System (GPSS V Program Product 5734-XS2) is a digital simulation program for conducting evaluations and experiments of systems, methods, processors and designs. The program provides many facilities to simplify simulation programming and to produce output reports. The program is a modification and adaptation of GPSS V available on the 360-370 Operating System.

## Restrictions - GPSS

The following special features of GPSS may give problems under MUSIC, and their use should be avoided if possible:

- Update
- Read/Save
- Jobtape
- Run Length
- Auxiliary Storage
- Load
- HELP routines

## Usage - GPSS

GPSS is invoked by using the /LOAD GPSS statement in the input stream. The /LOAD is followed by an optional /OPT statement specifying the main storage option (A, B, or C) to be used, then by the GPSS program statements. If /OPT is omitted, storage option A is used.

```
/LOAD GPSS
/OPT x                  (where x is A, B, or C)
...GPSS program statements...
```

The ddnames which are automatically predefined by the MUSIC/GPSS interface are: DINPUT1 (input statements), DOUTPUT (printed output), and the temporary work files DINTERO, DSYMTAB, DINTWORK, DREPTGEN, and DXREFDS. Other ddnames, if needed, must be defined by /FILE statements. If the default space is not enough for a temporary work file, supply an overriding /FILE statement of the form:

```
/FILE ddname NAME(&&TEMP) NEW RECFM(V) SPACE(n)
```

## Reference - GPSS

*General Purpose Simulation System V User's Manual*, (GH20-0851)

# Example - GPSS

```
/LOAD GPSS
*     GPSS HARBOUR SIMULATION PROBLEM
      SIMULATE
      GENERATE    2,1
      QUEUE 1
      ENTER 11
      DEPART      1
      QUEUE 2
      ENTER 12
      DEPART      2
      ADVANCE     15,6
      LEAVE 12
      TRANSFER    .9,,LVE
      ADVANCE     2,1
LVE   LEAVE 11
      TABULATE    10
      TERMINATE   1
11    STORAGE     6
12    STORAGE     5
10    TABLE M1,0,10,20
      START 100
      END
```

# Graphical Data Display Manager - GDDM

MUSIC supports the GDDM series of programs, providing presentation services in host computers. It drives displays, printers, plotters, and scanners, and includes several easy to use utilities for end-users. These utilities can be accessed via a full-screen menu.

As well, GDDM has a powerful and versatile programming interface. This means that from your high level language program (VS FORTRAN, PL/1, VS PASCAL, etc.) you can call GDDM subroutines and perform graphics function upon your workstation. Using PCLK, you can generate graphics on MUSIC/SP that can be displayed on your PC (when dialed into the MUSIC/SP host running on a 9370).

## Command - GDDM

To start any of the GDDM functions (Presentation Graphics, View Utility) issue the command: GDDM. From the panel screen, select the option and press enter. In order to use the System Programming Interface of GDDM, refer to the GDDM reference manual for the subroutine calling sequences. You must use the appropriate subroutine library as described below.

You should specify a 3 megabyte user region when using GDDM.

## Usage Notes - GDDM

If you are coding your own applications to use GDDM's System Programming Interface, then you will need to use the correct subroutine library when Linkediting the program.

If the program is using the GDDM reentrant interface, use the following subroutine library search order: (*GDDMRLIB, xxx, *GDDM, *OS, *MUS).

If your program is using the non-reentrant interface (this is usually the case), then use the following for the subroutine library search order: (*GDDMNLIB, xxx, *GDDM, *OS, *MUS)

In both of the above cases, replace "xxx" by *PGF, *IMD, *IVU when using the Presentation Graphics Facility, the Interactive Map Definition utility, or the Interactive View Utility, respectively.

For example, suppose that you are using the non-reentrant version of the System Programming Interface, in conjunction with the Presentation Graphics Facility. In generating a load module using the Linkage Editor, the required SUBLIBOS file statement would be:

```
/FILE SUBLIBOS PDS(*GDDMNLIB,*PGF,*GDDM,*OS,*MUS) SHR
```

placed prior to the /LOAD LKED statement.

Access to GDDM on MUSIC/SP via FTTERM V2, via the 9370 ASCII subsystem is supported. Consult the FTTERM documentation for further information.

**Restrictions**

1. You cannot use the MUSIC Loader with any GDDM applications. The MUSIC Linkage Editor must be utilized. This means that you must specify /JOB NOGO,DECK for the assembly or compilation step, and run a second job that performs the linkedit. (Refer to the Usage Notes above to see how to specify the GDDM subroutine libraries.)

2. If you are using the System Programming Interface of GDDM, then you MUST add the following call BEFORE attempting to call any GDDM facilities.

   CALL GDDMSB

   This call sets up the MUSIC-GDDM interface and then returns to the caller.

3. The sample User Tasking application requires a large amount of storage. Based on the order that you select the start-up, you can receive the MUSIC message about Insufficient Main Storage.

4. Depending on the type of workstation that you are using, the MUSIC multi-session facility may function a little differently. When you press the PA2 key, you MAY get a "X PROG752" message at the bottom of your workstation. This is caused by the fact that GDDM has set the workstation in 16 bit addressing mode. Multi-session is STILL available. You will NOT receive the message "Press function key to go to next session". To use multi-session, you must:
   1. Press RESET on the workstation.
   2. Press the function key for the multi-session function you wish to perform.

# Full-Screen Menu - GDDM

The following menu is presented when you issue the GDDM command from the *Go mode. This front-end menu allows you to start up the various GDDM utilities without having to remember the actual utility name.

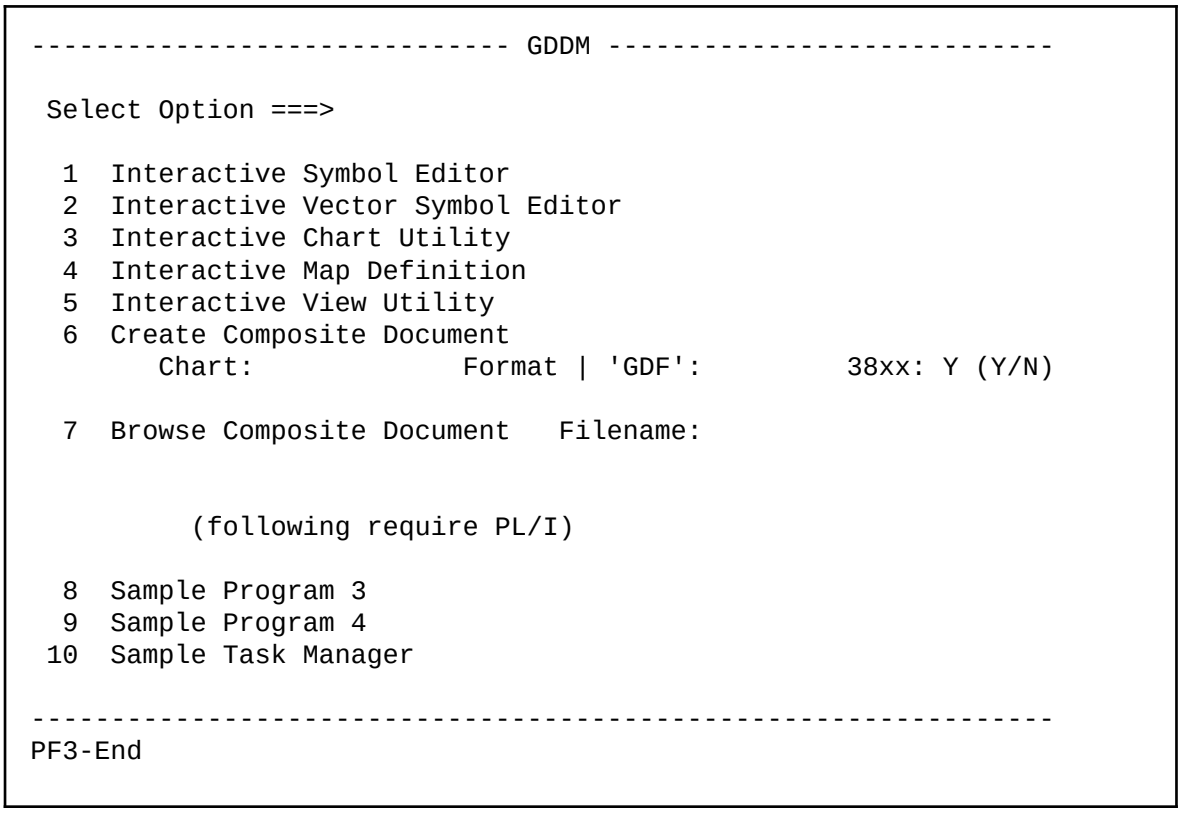



*Figure 8.7 - Menu for GDDM*

Two symbol editors (the Image Symbol Editor and the Vector Symbol Editor) allow you produce or modify symbols for annotating graphical output.

The Interactive Chart Utility helps you to draw charts in a simple manner for display on a screen or for printing. No programming knowledge is required in order to run the Interactive Chart Utility.

The Image Map Definition utility allows you to define the layout of the data that your application program presents on a display or output device. Basically, it allows you to define screens for your programs using GDDM.

The Interactive View Utility is an interactive program which allows users to create images by scanning documents, save images on disk, display them on screens, edit them in various ways, and create image output files for printers.

A composite document is a file containing text, images, and graphics. Option 6 on the GDDM menu allows you to create these documents and save them in MUSIC files. Option 7 enables you to view these documents on a workstation or a PC (using PCLK).

# References - GDDM

There are a large number of manuals available for GDDM. Refer to the General Information Manual below for more information on the manuals that you should reference. A selection of some other GDDM manuals has also been included below.

*GDDM General Information Manual*, (GC33-0319).

*GDDM Image View Utility*, (SC33-0479).

*GDDM Interactive Map Definition*, (SC33-0338).

*GDDM-PGF Interactive Chart Utility*, (SC33-0328).

*GDDM Image Symbol Editor*, (SC33-0329).

*GDDM-PGF Vector Symbol Editor*, (SC33-0330).

*GDDM Application Programming Guide*, (SC33-0337).

*GDDM-PGF Programming Reference*, (SC33-0333).

*GDDM PCLK Guide Version 1.1*, (part number 6242915).

## Examples - GDDM

The following samples show the MUSIC control statements needed to compile, linkedit and execute a GDDM sample program.

1. Compiling a GDDM Sample Program

```
/SYS REG=1000
/FILE SYSPUNCH NAME(PLI4.SAMPLE.OBJ) NEW(REPL)
/FILE SYSLIB PDS($GDM:*.M) SHR
/LOAD PLI
/JOB NOGO
/OPT INCLUDE,MARGINS(2,72,0),DECK
/INC $GDM:ADMUSP4.S
```

2. Link-Editing a GDDM Sample Program

```
/FILE LMOD NAME(PLI4.SAMPLE.LMOD) NEW(REPL) LR(128) RECFM(F)
/FILE SUBLIBOS PDS(*OS,*GDDMNLIB,*GDDM,*MUS) SHR
/LOAD LKED
/JOB NOGO,MODE=OS,MAP
/INC PLI4.SAMPLE.OBJ
```

3. Executing a GDDM Sample Program

```
/SYS REG=3000
/FILE ADMSYMBL PDS(*.SYM,$GDM:*.SYM) SHR
/FILE ADMDEFS N(GDDM.ADMDEFS) SHR
/FILE ADMGDF PDS(*.GDF) SHR
/COM if you have PC-Link then you need the next statement
/FILE ADMPC PDS($PLK:*.SYM) SHR
/FILE 3 UDS($GDMGDDM) VOL(MUSIC1) SHR
/FILE LMOD NAME(PLI4.SAMPLE.LMOD)
/LOAD XMPLI
TEMPNAME LMOD
```

# Linkage Editor - LKED

The MUSIC Linkage Editor is a loader which is capable of creating a load module with an overlay structure, similar to the IBM Operating System Linkage Editor. (The load modules, though, are not interchangeable with OS.) If an appropriate /FILE statement for a UDS or file is provided, the load module will be written on it. Load module data sets should be allocated with a record size of 128 bytes, and record format F should be used for files. When loading is complete, the job is executed (unless /JOB NOGO is specified). The Linkage Editor overlay supervisor is part of the user program during execution and requires about 2000 bytes of main storage.

Since load modules in some cases contain system-dependent coding, users should be prepared to recreate them if required because of system modifications.

*Notes:*

1. The ALIAS, INCLUDE and LIBRARY Operating System Linkage Editor control statements are not supported.

2. The user should note that if an overlay segment is to be used more than once, it must be serially reusable. Serially reusable means that an overlayed subroutine does not assume that variables will still contain values set by some previous execution of that subroutine.

3. If a program to be overlaid uses VS FORTRAN direct access input/output, all DEFINE FILE statements must be in the root segment of the overlay.

## Link-editing COBOL, VS Assembler and PL/I Programs

VS FORTRAN, COBOL, VS Assembler, and PL/I programs which perform input/output through BSAM or QSAM macro instructions must be link-edited with MODE=OS and NOGO specified on the /JOB statement. To run these programs from a load module, the user must use the /LOAD XMON statement (for COBOL, VS FORTRAN, and VS Assembler) or the /LOAD XMPLI statement (for PL/I) in place of the /LOAD EXEC statement.

## Usage - LKED

The Linkage Editor is invoked by a /LOAD LKED statement. This statement may be followed by object modules and overlay control lines or by a /OPT SYSIN=n pointing to a data set containing object modules and overlay control statements.

The file to contain the load module is defined by a /FILE statement specifying ddname LMOD. This /FILE statement precedes the /LOAD LKED statement. Unit number 3 may be used instead of ddname LMOD. Either a file or a UDS may be used, as in the following examples:

```
/FILE LMOD NAME(filename) NEW LRECL(128) RECFM(F) SPACE(40)

/FILE LMOD UDS(dsname) VOL(MUSIC1) NEW LRECL(128) NREC(500)
```

To distinguish load module files from other files, it is a good idea to use a suffix such as .LMOD at the end of the file name.

# Parameters - LKED

The /LOAD LKED statement may be followed by a /JOB line indicating optional parameters.

```
/JOB  [NOMAP][,NOCALL][,NONOPRINT][,NOGO][,SEQUENCE][,TRACE]
      [MAP  ][,CALL  ][,NOPRINT  ][,GO  ]
                      [,PRINT    ]

           [,NAME=prgnm][,MODE=OS][,LET][,STATS][,MAXGAP=n]
```

NOMAP No module map to be produced. This is the default for workstation jobs.

MAP A module map showing each csect and entry point with its assigned relative address is to be printed. This is the default for batch jobs. Absolute addresses can be found by adding hexadecimal 4A00 to the relative addresses.

NOCALL Do not search the system subroutine library for required modules not present in the input stream. The parameter NOSEARCH is equivalent to NOCALL.

CALL Search the system subroutine library for required modules not present in the input stream. This is the default. The parameter SEARCH is equivalent to CALL.

NONOPRINT Suppress printing of all control statements and error messages.

NOPRINT Suppress printing of all control statements. This is the default for workstation jobs.

NOLIST Same as NOPRINT.

PRINT Print all control statements and error messages. This is the default for batch jobs.

LIST Same as PRINT.

NOGO Do not execute the load module produced.

GO Execute the load module only if no errors were detected during Linkage Editing.

LET Execute the load module even if errors were detected during Linkage Editing. This is the default.

SEQUENCE Check sequence numbers in columns 77-80 of all input object modules to detect any missing or out-of-sequence lines. Blank sequence numbers are ignored.

TRACE Display a message identifying each csect as it is read by the Linkage Editor.

MAXGAP=n Specifies that any uninitialized storage areas of length greater than *n* bytes are to be represented as a gap in the load module text on disk. The default is MAXGAP=8192.

The following parameters are only meaningful if the load module is being created on a file that is to be kept for later use.

NAME=prgnm Specifies the name of the load module, and is required. *prgnm* can be up to 8 characters in length and must be alphanumeric with no special characters. The Linkage Editor control

statement NAME will override this option. If not specified, NAME=TEMPNAME is assumed.

STATS — Specifies that the Linkage Editor is to display a message showing the total number of blocks in the load module data set, and the number of blocks used for this load module. A *block* is a 512 byte piece of the file. Thus the file statement "/FILE LMOD UDS(&&TEMP) NREC(400) RSIZ(128)" contains 100 blocks.

MODE=OS — Specified if link-editing COBOL, PL/I or VS Assembler programs as described above. The NOGO option must also be specified.

*Notes:*

1. If a /JOB statement is used, it should appear immediately after the /LOAD LKED statement. More than one /JOB line may be used if necessary, but each parameter must be wholly contained on one /JOB line.

2. For a very large program, the Linkage Editor may stop with an error message such as `too many RLDs` or `too much text.` In that case, try the Linkage Editor job again in a larger user region, e.g. /SYS REGION=1024. A larger work file may also be needed (see note 3 below).

3. The Linkage Editor uses a temporary UDS with a DDNAME of LKEDWORK as a work file. This UDS has a default size of 8000 128-byte records. If needed, the user can increase this size by specifying a permanent UDS (which is deleted at the end of the job) by using the appropriate /FILE statement. For example:

```
/FILE LKEDWORK UDS(dsname) NREC(20000)
/ETC LRECL(128) VOL(MUSIC1) NEW DELETE
```

4. The following statement is needed when you use /LOAD LKED to execute a COBOL2 load module:

```
/FILE SUBLIBOS PDS(*COBOL2,*MUS,*OS,*EXT)
```

# Control Statements - LKED

The Linkage Editor control statements normally begin at or after column 2. These statements should be preceded by all object modules (or /INCLUDE statements pointing to object module files).

Note that some Linkage Editor control statements have names which are similar to MUSIC's statements though they are NOT the same. (The MUSIC Linkage Editor takes its control statement names from those of the OS Linkage Editor.)

The following Operating System Linkage Editor control statements are supported on the MUSIC Linkage Editor:

OVERLAY
INSERT
ENTRY
NAME
REPLACE
CHANGE

The following control statements are not supported:

ALIAS
INCLUDE
LIBRARY

The /OPT SYSIN=n control statement may be used to read the input from MUSIC unit number n. Normally this control statement is used when you want to read object modules from a UDS file. When end-of-file or a /DATA statement is reached on the data set the Linkage Editor switches back to the normal input stream for further input.

## Linkage Editor Return Codes

The following job exit codes (return codes) are set by the Linkage Editor:

0 No errors or warnings.
4 Some warning messages were issued. For example, there were some unresolved external references.
8 Some error messages were issued.
16 The Linkage Editor job was aborted because of a serious error.
>16 Some other system error.

If more than one load module member is created in the job, i.e. more than one NAME statement was used, the exit code is the highest code encountered for all the members.

If /JOB NOGO is not used, and the user's program is executed as part of the Linkage Editor job, the exit code is as set by the user program, rather than as above.

## Reference - LKED

*IBM 360/370 Operating System Linkage Editor and Loader* (GC28-6538)

## Example - LKED

```
/FILE LMOD NAME(PROG2.LMOD) NEW LRECL(128) RECFM(F)
/LOAD LKED
/JOB MAP,NAME=TEST2
/INCLUDE MAIN     (FILE CONTAINING THE MAIN PROGRAM)
/INCLUDE SUBS     (FILE CONTAINING THE THREE SUBROUTINES)
    OVERLAY A
    INSERT SUB1
    OVERLAY A
    INSERT SUB2
    OVERLAY A
    INSERT SUB3
/DATA
    - USER DATA -
```

Please refer to the example in the description of "EXEC - LOAD MODULE EXECUTOR" in this chapter for the control statements used to execute the load module created above.

# Load Module Executor - EXEC

The Load Module Executor is used to load and execute a program which has previously been stored in a file by the MUSIC Linkage Editor. The executor is a part of the user program during execution, and requires about 2000 bytes of main storage. This loading process can be considerably faster than the use of the standard loader.

Since load modules in some cases contain system-dependent coding, users should be prepared to recreate them if required because of system modifications.

## COBOL, ASM and PL/I Load Modules

The /LOAD EXEC processor automatically brings in the FORTRAN G interface. You must use the /LOAD XMON procedure if the load module is produced by an object module from some other compiler. For a load module produced by an object module from a PL/I program, use the /LOAD XMPLI procedure. The OS/MUSIC interface will be loaded in these cases.

## Usage - EXEC

The Load Module Executor is invoked by a /LOAD EXEC statement. /LOAD EXEC specifies that a load module contained in a file is to be executed. /LOAD statements must not be used following this statement.

The /LOAD EXEC statement must be preceded by a /FILE statement specifying the file which contains the program, normally with a ddname of LMOD.

## Control Line - EXEC

The Load Module Executor requires a single control line, which must be placed immediately after the /LOAD EXEC statement. This line is followed immediately by any user data required. (The /DATA statement is not used).

The format of the control line is as follows:

```
membername      ddname
                 or
                unit number
```

The member name must start in column 1, and is the name specified in the NAME=xxx option or on the NAME control statement when the load module was created by /LOAD LKED.

The second item is the ddname or unit number used on the load module /FILE statement in the /LOAD EXEC job. The default is ddname LMOD. It is separated from the membername by 1 or more blanks or commas.

# Example - EXEC

Please refer to the example in the description of "LKED - LINKAGE EDITOR" in this chapter for the control statements used to create the load module referred in the following example. The load module was created in the file PROG2.LMOD, with module member name TEST2.

The following example shows how to cause this load module to be executed.

```
/FILE LMOD NAME(PROG2.LMOD)    (load module)
/LOAD EXEC
TEST2                          (member name of load module)
/INCLUDE DATA1                 (user's data)
```

*Note:* A /DATA statement does not precede user data.

# Load Module Executor - XMON

The /LOAD XMON processor allows programs to be executed from load modules. This usually results in faster loading (compared with /LOAD ASMLG or /LOAD COBLG) and allows overlay structures to be used. The load module is created from the program object modules in the usual way, using the MUSIC Linkage Editor, except that the parameters *MODE=OS* and *NOGO* should be specified on the Linkage Editor /JOB statement. The load module is executed using /LOAD XMON, in much the same way as /LOAD EXEC is used. For /LOAD XMON, the user's load module is loaded at address 4A00 (hexadecimal).

## Usage - XMON

```
/LOAD XMON
```

XMON is used as follows:

```
/FILE statements as required
/FILE LMOD NAME(filename) or /FILE LMOD UDS(dsname) VOL(volume)
/LOAD XMON
parameter statement (see below)
...user data if any...
```

The parameter statement (immediately following /LOAD XMON) has the following format:

| membername | ddname<br>or<br>unit number | options |
|---|---|---|

The member name must start in column 1, and is the name specified in the NAME= option or on the NAME control statement when the load module was created by /LOAD LKED.

The second item is the ddname or unit number used on the /FILE statement for the load module file. The default is ddname LMOD. This item must be specified if any additional options follow.

The following additional options, separated by blanks or commas, can be used:

SVCTRACE to trace supervisor call instructions in the program.

IOTRACE to trace input/output operations.

DUMP to produce a storage dump, in some cases, if the job abnormally terminates.

CDUMP conversational dump.

DEBUG invoke full-screen debugger.

NONRENT load a non-reentrant copy of OSTRAP into the user region. This enables you to test OSTRAP with DEBUG (requires that the debugged module reside in the user region).

# LOADER - System Loader

The MUSIC loader may be used to load all programs except overlays and those programs requiring the OS/MUSIC interface. An input stream which consists entirely of object modules produced by VS Assembler will be processed more efficiently if the loader processor is invoked directly. After loading is complete, the program is executed (unless /JOB NOGO is specified).

*Notes:*

1. Object modules from high level language compilers and VS Assembler programs using BSAM or QSAM macro instructions cannot be executed using this loader. They require either /LOAD COBLG, /LOAD PLILG, or /LOAD ASMLG.

2. The loader can handle a maximum of 255 control section names and entry point names. If the combined total of CSECT names and entry names exceeds 255, use the linkage editor.

## Usage - LOADER

The loader is invoked by the use of a /LOAD LOADER statement.

/LOAD LOADER specifies that the lines following consist exclusively of object modules to be loaded and executed by the MUSIC loader. Utilization of processing unit time is optimized if this statement is used instead of /LOAD BASIC or FORTG1. /LOAD statements following this statement are ignored.

## Parameters: Loader Step - LOADER

```
/JOB  [GO  ][,MAP  ][,XESD=sss][,XRLD=rrr][,NOPRINT]
      [NOGO][,NOMAP]
```

| | |
|---|---|
| GO | The program is to be loaded and executed. This is the default. |
| NOGO | The program is to be compiled only. (This option is often used when the DECK option on the /OPT statement is used from a workstation.) |
| MAP | A storage map showing where the user's programs and subprograms are loaded into main storage is to be produced. |
| NOMAP | No storage map is to be produced. |
| XESD=sss | The number of extra external symbol dictionary entries required (indicated when the loader is unable to load the program). |
| XRLD=rrr | The number of extra relocation dictionary entries required (indicated when the loader is unable to load the program). |
| NOPRINT | All loader messages, except error messages, are to be suppressed. |

*Note:* If a /JOB statement is followed by another /JOB statement, only the last one is processed.

```
/OPT [SYSIN=n]
```

SYSIN=n Specifies that input is to be taken from MUSIC unit number n (1 through 15). A /FILE statement defining MUSIC I/O unit n as the appropriate file must appear at the beginning of the job. When an end-of-file or a /DATA statement is encountered on this input data set, further input is taken from the normal input stream. If another /OPT SYSIN=n is found in the input stream, it will be processed in the same manner.

## Example - LOADER

In the following example an object module was saved in a file called SAMPLE.OBJ. Edit the file SAMPLE.OBJ, and add the Job Control statement /LOAD LOADER as the first line. Then file the object module.

The example below shows the object module being executed.

```
*Go
sample.obj
*In progress
003638 BYTES USED
EXECUTION BEGINS
?
25.
THE ROOT OF  25.00 IS   5.00
?
/cancel
*Terminated
*Go
```

# PASCAL Compiler - VSPASCAL

Programs written in the PASCAL language can be compiled and run using the IBM VS PASCAL compiler (Program Product 5668-767). Your installation may choose not to have this compiler available for your use.

VS PASCAL programs are normally compiled and executed using the OS/MUSIC interface. The VS PASCAL processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed.

*Notes:*

1. Since VS PASCAL programs can invoke many of the facilities of the Operating System, some of which are not available on MUSIC, it is possible to compile a VS PASCAL program which will not execute on MUSIC.

2. MUSIC supports fixed length sequential files. This includes VS PASCAL TEXT and RECORD file formats. VS PASCAL programs using the procedures PDSOUT and UPDATE (for sequential files) may be compiled but not run on MUSIC. Variable length records are supported only on output, and must not be blocked.

3. VS PASCAL direct access files using the SEEK procedure and fixed-length records are supported, provided that RESET or UPDATE procedure is used when opening the file. The REWRITE procedure must not be used for direct access files. For reading, open the file using the RESET procedure. For writing, use the UPDATE procedure. A new direct access file must be initialized in VS PASCAL by writing records **in order** (0,1,2,...) to the end of the dataset. Internally, each direct access record starts on a 512-byte block and occupies one or more blocks. To initialize a direct access file, use the REWRITE procedure to create a sequential file, writing valid or blank records in order to the file. Once the file has been created, it can be accessed as a direct access file.

4. If the VS PASCAL %INCLUDE statement is used, a /FILE statement with a ddname of SYSLIB and parameter PDS must be defined at the beginning of the job. (Consult the description of the /FILE statement for files.) By default, the system provides a SYSLIB file statement pointing to the 'standard' VS PASCAL 'include library. If you wish to provide your own library and still have access to the 'standard' list, you must use the following SYSLIB statement.

   ```
   /FILE SYSLIB PDS('your library specifications',$VP2:*.M) SHR
   ```

   VS PASCAL supports two different forms of the %INCLUDE statement. Under MUSIC, only the following form is supported.

   ```
   %INCLUDE member-name
   ```

   The MUSIC /INCLUDE statement may also be used to perform a function similar to the %INCLUDE statement.

## Usage - VSPASCAL

The VS PASCAL compiler is invoked by the use of a /LOAD VSPASCAL statement. /LOAD VSPASCAL specifies that the lines following consist of source statements to be processed by the IBM VS PASCAL compiler. Object modules may also be present. No other /LOAD statements should follow this one.

# Main Storage Requirements - VSPASCAL

If the compiler is not in MUSIC's Link Pack Area, a region of at least 512K is required. If the compiler is in the Link Pack Area, a 256K region is large enough for small programs.

# Interlanguage Communication - VSPASCAL

It is sometimes desirable to invoke subprograms written in other programming languages. A VS PASCAL procedure may call subroutines written in Assembler and FORTRAN. Subroutine written in COBOL or PL/I may not be called.

You may also invoke VS PASCAL procedures as subprograms from Assembler, COBOL and PL/I. When calling VS PASCAL subroutines from other languages, do not call the VS PASCAL procedure PSCLHX to cleanup the environment as documented. This procedure is not needed when running under MUSIC.

# External File Names - VSPASCAL

A standard set of external file names (ddnames) is provided, along with default values for logical record length (LRECL).

| DDNAME | LRECL | MUSIC I/O |
|---|---|---|
| INPUT | 80 | input stream (data following /DATA) |
| SYSPRINT | 133 | printed output |
| OUTPUT | 133 | printed output |
| SYSIN | 80 | conversational input (workstation job)<br>input stream (batch jobs) |

Additional ddnames can be defined by specifying /FILE statements with the corresponding ddnames at the beginning of the job. (Consult the description of the /FILE statement.)

For any file, the LRECL and BLKSIZE may be specified in the VS PASCAL program by the LRECL and BLKSIZE options of the RESET, REWRITE and UPDATE procedures.

End-of-file can be indicated on a conversational read (of a TEXT file) by typing /EOF.

# Parameters: Compiler Step - VSPASCAL

Compiler options may be specified on a /OPT statement preceding the VS PASCAL source. The options should be separated by commas, and the last option must be followed by a blank. A /OPT statement may also be used to indicate the start of a new procedure. If more than one /OPT statement is used, the effect is cumulative, that is, each option remains in effect for the rest of the job until reset by a later /OPT statement.

The default compiler options are listed below. Refer to the VS PASCAL Programmer's Guide for a description of the options and their abbreviations.

| Workstation Defaults: | Batch Defaults (if different): |
|---|---|
| CHECK | |
| NODEBUG | |
| GOSTMT | |
| LANGLVL(EXTENDED) | |
| LINECOUNT(61) | |
| NOLIST | |
| MARGINS(1,72) | |
| OPTIMIZE | |
| PAGEWIDTH(128) | |
| PXREF | |
| SEQUENCE(73,80) | |
| NOSOURCE | SOURCE |
| NOXREF | XREF(SHORT) |
| DDNAME(COMPAT) | |
| HEADER | |
| CONDPARM() | |
| NOGRAPHIC | |
| STDFLAG(E) | |

/FILE statements with ddnames of COMP.SYSIN, COMP.SYSPRINT and COMP.SYSLIN can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions.

*Creating Object Modules*

If you wish to compile a VS PASCAL procedure and save the object module for later execution, the following procedure should be followed.

1. Specify via the /FILE statement with ddname SYSLIN the name of the file to contain the object module.

2. Include the /JOB NOGO statement following the /LOAD statement. This will invoke the VS PASCAL compiler but will not executed the program after compilation.

VS PASCAL object modules can be executed using the OS-MODE loader, ASMLG. If a load module was created, then you must use the XMON procedure to execute the load module.

# Parameters: Loader Step - VSPASCAL

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,DUMP][,CDUMP][,DEBUG]
      [FULMAP]

          [,FILRFM][,NOSEARCH][,IOTRACE[(C)]][,SVCTRACE[(C)]]
```

Parameters can be separated by one or more commas or blanks.

MAP List a storage map of the loaded program.

| | |
|---|---|
| FULMAP | List an extended storage map of the loaded program. |
| NOGO | Do not load or execute the program. |
| LET | Allow the program to be executed even if unresolved external references (such as missing subroutines) are found. |
| NOPRINT | Informative and diagnostic messages are not to be produced. |
| DUMP | Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump. |
| CDUMP | Request a conversational trace and dump when an error occurs. |
| DEBUG | Load the DEBUG program into memory for debugging. |
| FILRFM | Supply the RECFM V if the file is V/VC. |
| NOSEARCH | Do not search the subroutine library for unresolved routines. |
| IOTRACE | Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed. |
| SVCTRACE | Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed. |

*Notes:*

1. Multiple /JOB statements may be used if desired.

2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

## Parameters: Go Step - VSPASCAL

Run time options can be specified on the "/PARM xxx" statement at the beginning of the job, where *xxx* is the option string. To distinguish run time options from the parameter string intended to be processed by the program, the options must proceed the parameter string (if any) and be terminated with a slash ("/").

SVC instructions and I/O operations can be traced at execution time. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement. The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the /JOB Control statement above for a description of the syntax.)

## Title Lines - VSPASCAL

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

## VS PASCAL Interactive Debugger

The VS PASCAL interactive debugger is available under the MUSIC system. In order to use the debugger, you must follow the following steps:

1. Compile the module to be debugged with the DEBUG option. Modules that have been compiled with the DEBUG option can be loaded with modules that have not been compiled with the DEBUG option.

2. Specify on the /PARM statement, the DEBUG run-time option. After the modules have been loaded, the debug environment will be active and you will be immediately prompted for debugger commands.

3. The ATTN or BREAK keys on the workstation can be used to signal the debug environment that you want to gain control. On a 3270 workstation, after you press PA1 the message **ATTN** will be displayed in the lower right corner of the workstation. Press the ENTER key to signal the interrupt to the debug environment.

4. Use the VS PASCAL interactive debugger after creating a load module via the linkage Editor.

5. Use of the debugger environment will add about 70K to the User Region needed to execute the program.

## References - VSPASCAL

*VS PASCAL Application Programming Guide*, (SC26-4319).

*VS PASCAL Language Reference*, (SC26-4320).

*VS PASCAL Reference Summary*, (SX26-3760).

## Examples - VSPASCAL

1. The following sample program uses READ and WRITE statements on record files.

```
/LOAD VSPASCAL
program Sample;

type
  REC = record
          NAME : STRING(25);
          AGE  : 0..99;
          SEX  : (MALE,FEMALE)
        end;
var
  INFILE,
  OUTFILE:
      file of REC;
  BUFFER : REC;
begin
  RESET(INFILE);
  REWRITE(OUTFILE);
  while not EOF(INFILE) do
    begin
```

```
      READ(INFILE,BUFFER);
      WRITE(OUTFILE,BUFFER)
    end
end.
```

2. The following VS PASCAL program will be compiled and the interactive debug environment will be entered once the program has been loaded. Refer to the "Debug Terminal Session" in the *VS PASCAL Programmer's Guide* for an example of a debug session.

```
/SYS REG=340
/PARM DEBUG/
/LOAD VSPASCAL
/OPT DEBUG
program Debugging;
   var A, B : REAL;
begin
   A := 1;
   B := 2;
   Writeln ('A =     ', A);
   Writeln ('A + B = ', A+B);
end.
```

*Note:* The /OPT statement specifies the DEBUG compiler option. The /PARM statement specifies the DEBUG routine option. Both are required for DEBUG to be entered. In the /PARM statement, the trailing "/" serves to delimit VS PASCAL routine options from parameters that you might want to pass to your PASCAL program.

# PL/I Version 2 Optimizing Compiler - PLI

Programs written in the PL/I language can be compiled and run using the IBM OS PL/I Version 2 Optimizing compiler (Program Product 5668-909). This compiler is compatible with previous releases of Version 2 and Version 1 (existing programs will work with this release).

The PL/I processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed.

## Usage Notes - PLI

Since PL/I programs can invoke many of the facilities of the MVS operating system, some of which are not available on MUSIC, it is possible to compile a PL/I program which will not execute on MUSIC.

1. MUSIC supports fixed length sequential files. This includes PL/I GET and PUT statements, and READ and WRITE statements for sequential files. The UPDATE and SEQUENTIAL attributes must not be used together. Variable length records are supported only on output, and must not be blocked.

2. Direct access files using REGIONAL(1) and fixed-length records are supported, provided that the UPDATE or INPUT attribute is used on the file declaration statement. The OUTPUT attribute must not be used. For reading, use the READ FILE statement. For writing, use the WRITE FILE or REWRITE FILE statement. A new direct access file should be initialized in PL/I by writing records **in order** (0,1,2,...) to the end of the data set. Internally, each direct access record starts a new 512-byte block, and occupies one or more blocks.

3. Sequential (ESDS), direct (RRDS), and indexed (KSDS) file access are available through MUSIC's VSAM support. For information on creating and using VSAM files with PL/I, refer to the topic "MUSIC/SP VSAM" in *Chapter 4. File System and I/O Interface*

   ISAM is not supported.

4. If the PL/I %INCLUDE statement is used, a /FILE statement with a ddname of SYSLIB and parameter PDS must be defined at the beginning of the job. (Consult the description of the /FILE statement for files.) The compiler option MACRO or INCLUDE must also be specified on the /OPT statement. The MUSIC /INCLUDE statement may also be used to perform a function similar to the %INCLUDE statement.

5. PL/I sort facilities are supported as described below. Extended precision arithmetic (up to 32 significant decimal digits) is supported.

6. The following are not supported:

   Multi-tasking.
   Teleprocessing (TRANSIENT file attribute).
   Execution-time COUNT, FLOW and REPORT options.
   Unbuffered attribute for RECORD files.
   Regional(2) and Regional(3) files

# Usage - PLI

The OS PL/I Optimizing Compiler is invoked by the use of a /LOAD PLI statement. This processor invokes the PL/I Optimizing Compiler, and then the loader. After loading, the program is automatically executed using the OS/MUSIC interface (unless /JOB NOGO is used or the compiler return code is 12 or more). The /OPT and /JOB control statements may be used with this processor. PL/I object modules can optionally be produced and saved, intermixed with source. The object modules are identical with those produced on OS.

# Available Unit Numbers and Buffer Space - PLI

Any tape blocksize up to 32760 may be used, provided the user region size (specified on /SYS REGION=nnn) is large enough.

The user can use 3 UDS files. MUSIC I/O unit 4 cannot be defined in /FILE statements since it is used for the compiler modules.

# External File Names - PLI

A standard set of external file names (ddnames) is provided, along with default values for logical record length (LRECL).

| DDNAME | LRECL | MUSIC I/O |
|---|---|---|
| SYSIN | 80 | input stream (data following /DATA) |
| SYSPRINT | 121 | printed output |
| SYSPUNCH | 80 | output to holding file (workstation jobs)<br>punched output (batch jobs) |
| SYSTERM | 121 | printed output |
| CONSOLE | 80 | conversational input (workstation jobs)<br>input stream (batch jobs) |

Additional ddnames can be defined by specifying /FILE statements with the corresponding ddnames at the beginning of the job. (Consult the description of the /FILE statement.)

For any file, the LRECL and BLKSIZE may be specified in the PL/I program by the RECSIZE and BLKSIZE options of the ENVIRONMENT attribute.

File names within PL/I should not exceed 7 characters. However, the TITLE option on the OPEN statement may be used to associate a ddname with an internal file name. For example:

```
OPEN FILE(SYSPUN) TITLE('SYSPUNCH') OUTPUT;
```

When doing a conversational input on a workstation, always use PUT SKIP to force the output in the output buffer to be displayed on the workstation before the conversational read is performed. Otherwise, the prompts, if any, will not be synchronized with the conversational reads.

End-of-file can be indicated on a conversational read by typing /EOF.

# Parameters: Compiler Step - PLI

Compiler options may be specified on a /OPT statement preceding the PL/I source. The options should be separated by commas, and the last option must be followed by a blank. A /OPT statement may also be used to indicate the start of a new external procedure. If more than one /OPT statement is used, the effect is cumulative, that is, each option remains in effect for the rest of the job until reset by a later /OPT statement.

The PL/I statement *PROCESS may be used in place of a /OPT statement. On a *PROCESS statement, options are separated by commas or blanks and the last option is followed by a semicolon. Options on *PROCESS statements are not cumulative.

The default compiler options are listed below. Refer to the *PL/I Optimizing Compiler Programmer's Guide* (*SC33-0006-5*) for a description of the options and their abbreviations.

| Workstation Defaults: | Batch Defaults (if different): |
|---|---|
| GOSTMT | |
| GRAPHIC | NOGRAPHIC |
| LMESSAGE | |
| OBJECT | |
| STMT | |
| NOAGGREGATE | |
| NOATTRIBUTES | |
| NOCOUNT | |
| NODECK | |
| NOESD | |
| NOFLOW | |
| NOGONUMBER | |
| NOIMPRECISE | |
| NOINCLUDE | |
| NOINTERRUPT | |
| NOINSOURCE | INSOURCE |
| NOLIST | |
| NOMACRO | |
| NOMAP | |
| NOMARGINI | |
| NOMDECK | |
| NONEST | |
| NONUMBER | |
| NOOFFSET | |
| NOOPTIMIZE | |
| NOOPTIONS | OPTIONS |
| NOSEQUENCE | |
| NOSOURCE | SOURCE |
| NOSTORAGE | |
| NOTEST | |
| NOXREF | |
| NOCOMPILE(S) | |
| CMPAT(V2) | |
| FLAG(W) | FLAG(I) |
| LINECOUNT(55) | |
| MARGINS(1,80,0) | |
| SIZE(MAX) | |
| NOSYNTAX(S) | |
| SYSTEM(MVS) | |

TERMINAL

/FILE statements with ddnames of PLI.SYSIN, PLI.SYSPRINT, and PLI.SYSPUNCH can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions. To provide more space to the default compiler work file SYSUT1 (default space is 300K) for very large programs, supply the following /FILE statement before /LOAD PLI:

```
/FILE SYSUT1 NAME(&&TEMP) RECFM(F) LRECL(4096) SPACE(550) NEW DELETE
```

# Parameters: Loader Step - PLI

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,DUMP][,CDUMP][,DEBUG]
      [FULMAP]

         [,FILRFM][,NOSEARCH][,IOTRACE[(C)]][,SVCTRACE[(C)]]
```

Parameters can be separated by one or more commas or blanks.

| | |
|---|---|
| MAP | List a storage map of the loaded program. |
| FULMAP | List an extended storage map of the loaded program. |
| NOGO | Do not load or execute the program. |
| LET | Allow the program to be executed even if unresolved external references (such as missing subroutines) are found. |
| NOPRINT | Informative and diagnostic messages are not to be produced. |
| DUMP | Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump. |
| CDUMP | Request a conversational trace and dump when an error occurs. |
| DEBUG | Load the DEBUG program into memory for debugging. |
| FILRFM | Supply the RECFM V if the file is V/VC. |
| NOSEARCH | Do not search the subroutine library for unresolved routines. |
| IOTRACE | Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed. |
| SVCTRACE | Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed. |

*Notes:*

1. Multiple /JOB statements may be used if desired.

2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. other OS-mode processors that do not invoke the OS-mode loader directly.

## Parameters: Go Step - PLI

Execution-time options can be specified by defining a PLIXOPT character string in the source program, or by specifying a "/PARM xxx" statement at the beginning of the job, where *xxx* is the option desired. The default execution-time options are:

HEAP(4K,4K)
ISAINC(0,0)
ISASIZE(x) where x=(region size - program size)/2
LANGUAGE(UENGLISH)
NOCOUNT
NOFLOW
NOREPORT
NOTEST
STAE
SPIE
TASKHEAP(4K,4K)

SVC instructions and I/O operations can be traced at execution time. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the /JOB statement for a description of the syntax.)

## Title Lines - PLI

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

## 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.

On MUSIC the sort is performed by a version of DSORT, the generalized sort routine described in *Chapter 10. Utilities* of this manual. The restrictions which apply to DSORT also apply to PL/I sorting. In particular, variable length records may not be sorted, sort control fields must begin on a byte boundary and be a whole number of bytes long, they may not overlap, and the maximum number of control fields is 20.

The argument specifying the amount of main storage for the sort must be at least 2048, and defaults to 32000 if not specified.

The following ddnames may be used. These ddnames, if required, must be defined using /FILE statements as described above in the section *External File Names*. An optional argument in the call to the sort interface routine may be used to change the first four characters *SORT* of the ddnames.

SORTIN Sort input if no E15 exit routine is defined.

SORTOUT Sort output if no E35 exit routine is defined.

SORTWK01 DSORT work file (optional).

SORTWK02 Second DSORT work file (optional). If SORTWK02 is not used, then SORTWK01 (if used) must be a UDS file, not a file.

If neither SORTWK01 nor SORTWK02 is defined by a /FILE statement, the sort must be small enough to be done entirely in main storage.

## References - PLI

*IBM OS PL/I Version 2 Programming Guide*, Release 2, (SC26-4307).

*IBM OS PL/I Version 2 Programming*: *Language Reference*, Release 2, (SC26-4308).

*IBM OS PL/I Version 2 Programming*: *Messages and Codes*, Release 2, (SC26-4309).

*IBM OS PL/I Version 2 Programming*: *Using PLITEST*, Release 2, (SC26-4310).

*IBM OS PL/I Version 2 Problem Determination*, Release 2, (LY27-9528).

## Example - PLI

This program reads numbers conversationally from the workstation and writes them to the data set FILE01. The end is indicated by typing /EOF. The object module is written to the file COPY.OBJ.

```
/FILE SYSPUNCH NAME(COPY.OBJ) NEW(REPLACE)
/FILE FILE01 UDS(USERFILE) VOL(MUSIC2) OLD
/LOAD PLI
/OPT DECK
COPY: PROC OPTIONS(MAIN);
ON ENDFILE(CONSOLE) GO TO FINISH;
LOOP: GET FILE(CONSOLE) LIST(X);
      PUT FILE(FILE01) LIST(X);
      GO TO LOOP;
FINISH: CLOSE FILE(FILE01);
END COPY;
```

## PLITEST

PLITEST is a flexible and efficient tool for examining, monitoring, and controlling program execution. You can use PLITEST interactively or in batch mode.

PLITEST is easy to learn and use. You can begin testing with PLITEST after learning just a few concepts: how to invoke PLITEST, and how to set, display, and remove breakpoints. PLITEST commands are similar to PL/I statements, so you know most of the commands already. And, online help is available at your finger-tips when using PLITEST in an interactive mode.

The IBM publication *OS PL/I Version 2 Programming*: *Using PLITEST*, (SC26-4310-1) describes the how to use PLITEST. It tells you how to select the various testing options that are available, how to set breakpoints, how to inspect and modify variables, and is a complete reference for PLITEST commands. The information below pertains to the specific usage of PLITEST on MUSIC.

On MUSIC only "Line Mode" operation is supported. To use PLITEST you must compile the program with the TEST Compiler Option. You specify this on the /OPT or *PROCESS statement. This option causes the compiler to include the appropriate "hooks" in the generated code to allow subsequent testing of the program. A number of subparameters can be specified on the TEST option that specify what type of testing is to be performed.

The TEST Run-Time Option should also be specified on the /PARM statement when you want to actually test the program. This also allows you to specify a file that contains the PLITEST commands. If none is specified the commands are read from the workstation.

PLITEST keeps a log file of the test session in the file pointed to by the ddname PLILOG. A file statement for this must be included in the test job, even if it is only a dummy file. The log file can be used as input to subsequent test runs. For example, suppose after a long interactive testing session you discover a problem and have to go back to fix it in the source. You can use the commands in the log file to quickly bring you back to where you were before and proceed to debug from that point.

To get online help while in interactive mode, type HELP when prompted for a PLITEST command.

## Sample PLITEST session

The following is a listing of the file used in the sample PLITEST session.

```
 /SYS REG=1024
 /FILE PLILOG DUMMY
 /PARM LANGUAGE(EN),TEST
 /LOAD PLI
/OPT SOURCE,TEST(ALL)
  /* PLI TEST PROGRAM */
 TESTP: PROCEDURE OPTIONS(MAIN);
 ON ENDFILE(CONSOLE) GO TO FINISH;

 PUT SKIP LIST ('THIS PROGRAM CALCULATES THE SQUARES OF NUMBERS');

 /* LOOP TO READ INPUT */
 LOOP:  PUT SKIP LIST ('ENTER A NUMBER');
        PUT SKIP;
        GET FILE(CONSOLE) LIST(X);
        Y=X**2;
        PUT SKIP LIST(X,' SQUARED IS ',Y);
        GO TO LOOP;

 /* ALL DONE */
 FINISH: PUT SKIP LIST('ALL DONE');
         END;
```

In the following sample PLITEST session the commands typed in by the user are preceded by the ">" character.

```
>pli.test
 5668-910 IBM OS PL/I OPTIMIZING COMPILER V2.R2.M1    11 DEC 89   12:04
 SOURCE,TEST(ALL);
                    SOURCE LISTING
     STMT
           /*  PL/I TEST PROGRAM  */
        1 TESTP: PROCEDURE OPTIONS(MAIN);
        2 ON ENDFILE(CONSOLE) GO TO FINISH;
        3 PUT SKIP LIST ('THIS PROGRAM CALCULATES THE SQUARES OF NUMBER
          /* LOOP TO READ INPUT */
        4 LOOP:  PUT SKIP LIST ('ENTER A NUMBER');
        5        PUT SKIP;
        6        GET FILE(CONSOLE) LIST(X);
        7        Y=X**2;
        8        PUT SKIP LIST(X,' SQUARED IS ',Y);
        9        GO TO LOOP;
          /* ALL DONE */
       10 FINISH: PUT SKIP LIST('ALL DONE');
       11         END;
 END OF COMPILATION OF TESTP
 00B130 BYTES USED
 EXECUTION BEGINS
 TEST:
>QUERY PROGRAM
 The compile-time data for program TESTP is
     The statement table has the STMT format.
     The program was compiled with the following options:
              TEST(ALL,SYM)
              NOOPTIMIZE
              NOGRAPHIC
              NOINTERRUPT
              CMPAT(V2)
              SYSTEM(MVS)
     This program has no subblocks.
 TEST:
>AT 9
 TEST:
>GO
 THIS PROGRAM CALCULATES THE SQUARES OF NUMBERS
 ENTER A NUMBER
>5
 TEST (TESTP:9):
>LIST(X,Y)
   5.00000E+00
   2.50000E+01
 TEST:
>GO
   5.00000E+00             SQUARED IS              2.50000E+01
 ENTER A NUMBER
>8
 TEST (TESTP:9):
>LIST(X,Y)
   8.00000E+00
```

```
  6.40000E+01
 TEST:
>GO
  8.00000E+00             SQUARED IS              6.40000E+01
 ENTER A NUMBER
>/EOF
 You have been prompted because the ENDFILE ( CONSOLE ) condition
 has been raised in your program.
 The current location is TESTP : 6.
 TEST:
>GO
 You have been prompted because the FINISH condition has been raised in
 has been raised in your program.
 The current location is TESTP : 11.
 TEST:
>GO
 ALL DONE
```

# PL/I OS-Mode Loader - PLILG

An input stream that consists entirely of object modules produced by the PL/I Optimizing Compiler will be processed more efficiently if the /LOAD PLILG statement is used to inform the system that this is the case.

## Usage - PLILG

This loader is invoked by the use of a /LOAD PLILG statement. It loads and executes a PL/I program in object module form. It is more efficient to use this processor than /LOAD PLI when the input consists solely of object modules. The /JOB control statement may be used with this processor to specify LOADER options.

The statement is used a follows:

```
/FILE statements as required
/LOAD PLILG
...object modules produced by the PL/I Optimizing Compiler
```

## Available Unit Numbers and Buffer Spaces - PLILG

The user can use 3 UDS files. MUSIC I/O unit 4 cannot be defined in /FILE statements since it is used for the PL/I transient library modules, which are loaded dynamically during execution.

## Parameters - PLILG

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,NOSEARCH][,FILRFM]
      [FULMAP]
```

Parameters can be separated by one or more commas or blanks.

MAP List a storage map of the loaded program.

FULMAP List an extended storage map of the loaded program.

NOGO Do not load or execute the program.

LET Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.

NOPRINT Informative and diagnostic messages are not to be produced.

NOSEARCH Do not search the subroutine library for unresolved routines.

FILRFM Supplies a RECFM V if the file is V/VC.

*Notes:*

1. Multiple /JOB statements may be used if desired.

2. The DEBUG, CDUMP, DUMP, IOTRACE and SVCTRACE parameters for OS-mode processors are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

```
/OPT  [SYSIN=n]
```

SYSIN=n Specifies that input is to be taken from the MUSIC unit number n. A /FILE statement defining MUSIC I/O unit n as the appropriate input file must appear at the beginning of the job. When an end-of-file or a /DATA statement is encountered on this input data set, further input is taken from the normal input stream.

# PL/I Load Module Executor - XMPLI

The /LOAD XMPLI statement is used to load and execute a PL/I load module which has previously been stored on a User Data Set (UDS) file or file by the MUSIC Linkage Editor. This is similar to /LOAD XMON. It results in faster loading (compared with /LOAD PLILG) and allows overlay structures to be used.

## Usage - XMPLI

This executor is invoked by the use of a /LOAD XMPLI statement. Usage is the same as for /LOAD XMON except that the load module data set must not be on I/O unit number 4.

## Available Unit Numbers and Buffer Space - XMPLI

The user can use a maximum of 3 UDS files, one of which may be the data set containing the PL/I load module. MUSIC I/O unit 4 cannot be defined in /FILE statements since it is used for the PL/I transient library.

# Restructured Extended Executor - REXX

REXX (Restructured Extended Executor) is a high level language which allows the execution of MUSIC commands and programs from directly within the REXX program itself, and thus, enables the user to chain together sequences of MUSIC commands under program control. The language constructs offered by REXX lend themselves well to structured programming and the powerful functions for command parsing, character handling, and data conversion, make it an excellent tool in the interactive environment. The language is described in the IBM publications *VM/SP System Product Interpreter User's Guide* (SC24-5238) and *VM/SP System Product Interpreter Reference* (SC24-5239). The sections of these publications that describe the usage of REXX in the CMS environment do not generally apply under MUSIC.

MUSIC has a tutorial for learning REXX. Type "TUT" in *Go mode or select the "Tutorials" item for the FSI main menu.

The following describes the usage of REXX in the MUSIC environment.

## Invoking REXX

The file containing a REXX program should contain the following control statements.

```
/INCLUDE REXX
  . . . .
  REXX program statements
  . . . .
```

The program can be executed from MUSIC command mode by typing the name of the file containing the program followed by an optional parameter string. For example, if a REXX program was contained in the file REX1, typing:

```
REX1 ABCD
```

would execute the program. The parameter string ABCD would be available to the program via the PARSE ARG instruction or the *ARG* function.

If you find it necessary to use a larger region size for a Rexx program, you must create a private file called REXX, containing:

```
/SYS NOPRINT,REGION=nnnn
/LOAD REXX
```

For example, a larger region would be needed for FSI file management if a very large number of files is involved. The default (and minimum) region size for /LOAD REXX is 1024K.

## Executing MUSIC Commands - REXX

MUSIC commands can be executed by simply stating them as character strings, or expressions that result in character strings, at the appropriate location in the program. When the command is completed, control is returned to the REXX program at the statement after the one invoking the command. The variable *RC* is set to the value of the return code as set by the execution of the command.

**Examples:**

1. Using a character string.

```
'EDIT FILE1'
```

The Editor will be invoked to edit the file called FILE1. When the edit session is finished, control is returned to REXX.

2. Using an expression.

```
name='FILE1'
cmd='EDIT'
cmd name
```

when the line *cmd name* is evaluated, the result will be the character string 'EDIT FILE1' which is passed to MUSIC as a command. This gives the same result as example 1.

*Note:* Commands that are processed directly by MUSIC's workstation command scanner are currently not supported by REXX. These are /COMPRESS, /CTL, /DISCON, /EXEC, /NS, /PAUSE, /PROMPT, /REQUEST, /RUN, /STATUS, /TIME, /TEXT, /TABIN, /TABOUT, /USERS, /WINDOW.

# Communicating with the workstation - REXX

The SAY instruction is used to write data to the workstation. If the first character of the output string is a valid workstation carriage control character, it is treated as such. Input from the workstation can be requested via the PULL or the PARSE EXTERNAL instruction. PULL (or PARSE PULL) first looks for data from the STACK. If this is empty, the data is read from the workstation. (The STACK is described later on.) The PARSE EXTERNAL instruction always reads from the workstation. The following complete REXX program illustrates the use of SAY and PULL.

```
/INC REXX

  /* program to ask the user's name */
  /* and says hello                 */

    say ' Hi there, what is your name.'
    pull name
    say ' hello ' name
```

# Accessing the STACK - REXX

REXX maintains an internal stack which is accessed by the PUSH, QUEUE and PULL instructions. All REXX programs in a particular program chain have access to the same stack so data can be passed from one REXX program to another via the stack. It should also be noted that since any REXX program in the chain has access to the stack, they can also change it. This does not present a problem when all the programs in the chain have been designed to work together, but the execution of a command may invoke some REXX program, not planned for in the original design, which modifies the stack, causing errors when it returns. Thus, if a REXX program is to be callable from any other REXX program, care must be taken to preserve the contents of the stack. The MUSIO command can be used to transfer the contents of all or part of the stack to or from the MUSIC Save Library. The following summarizes commands and instructions for accessing the stack.

PULL - get item from top of stack.
PUSH - put an item at top of stack.
QUEUE - put an item at bottom of stack.

QUEUED() - function: returns number of items in stack.

MUSIO - command: transfers stack to and from MUSIC's Save Library.

# MUSIO Command - REXX

The MUSIO command provides the interface between REXX and the MUSIC file system. A specified number of records are transferred from the Save Library to the stack or vice versa. The variable RC is set to the MUSIC file system return code. The format of the MUSIO command is as follows....

```
MUSIO option filename num
```

option READ, WRITE, APPEND, CLOSE.

- READ — If required, the file is opened. The requested number of records are read from the file and placed at the end of the stack. If the records are longer than 255 bytes, they are truncated. Subsequent reads will continue at the next record in the file. If an end of file is encountered, RC is set to 1. In this case, the QUEUED function can be used to determine the number of records read. If an error occurs in either the open or reading of the file, the return code (RC) will be non-zero. Since the stack is kept in main storage, reading a large file in one MUSIO request could cause the REXX program to run out of storage.
- WRITE — If required, the file is opened. If the file does not exist, a new file is created. The requested number of records are PULLed from the stack and written to the file. Subsequent writes will continue after the last record written. If the record in the stack is longer than the record length of the file, the record is truncated. If errors occur opening or writing to the file, the return code (RC) will be non-zero.
- APPEND — Similar to WRITE except it causes the file to be opened for append only access.
- CLOSE — The file is closed. This can also be used to rewind a file for subsequent processing.

filename The name of the file to access.

num The number of records to transfer. The string ALL indicates that all records should be transferred. The default is 1.

The following program, which lists a file, shows the use of MUSIO, PULL, and the QUEUED function.

```
/INC REXX

/*   list file XXXXX at the workstation */

    'MUSIO READ XXXXX ALL'  /* read the file */
    nrec=queued()           /* get number of records read */
    do i = 1 to nrec        /* loop to print stack */
      pull rec
      say rec
    end
```

# EXEC Command - REXX

Not all programs on MUSIC are executed by one-line commands. More typically, a small file is created containing control statements such as /FILE, /LOAD, and /INCLUDE. This can easily be accomplished from a REXX program. The appropriate control statements are QUEUED on the stack, MUSIO writes the stack to a file, and the file is executed by specifying the file name as a command.

```
/INC REXX

   /* put job in file X1 and execute it */

   queue '/FILE 10 N(PROG.OBJ) NEW(REPL)'
   queue '/LOAD VSFORT'
   queue '/OPT DECK'
   queue '/INC PROG.S'
   'MUSIO WRITE X1 ALL'
   'X1'         /* execute program in X1 */
```

When writing REXX programs to be used by others, the programmer cannot simply choose an arbitrary file name such as X1 to contain the program, since the user might actually have a file called X1. In order to avoid this problem the EXEC command is provided to execute whatever program is in the stack. The stack is written to the the reserved file @EXEC.tcb (tcb- terminal id number) and the program in this file is then executed.

```
/INC REXX

   /* use EXEC command to execute a program from the stack */

   queue '/FILE 10 N(PROG.OBJ) NEW(REPL)'
   queue '/LOAD VSFORT'
   queue '/OPT DECK'
   queue '/INC PROG.S'
   'EXEC'       /* execute what is in the stack */
```

# REXLIB Command - REXX

The REXLIB command provides a direct interface to the MUSIC LIBRARY command, avoiding the overhead of scheduling LIBRARY as a separate job. In addition the output from REXLIB can go directly into the REXX stack. Output is never written to the workstation. The parameters are the same as for the standard LIBRARY command with the addition of the "Q" parameter which indicates that any output should be queued in the stack. If you want REXLIB to operate like the DIR command rather than the LIBRARY

command, use the DIR option, for example:

```
REXLIB * Q DIR
```

REXLIB sets the following return codes.

```
  0 - no errors
  1 - not enough memory
  4 - invalid file name specification
  8 - invalid character in file name
 12 - error in user id
 16 - not authorized to look at this library
 20 - invalid character in user id
 24 - invalid parameter
 28 - conflicting parameters
 32 - parameter specified twice
 36 - invalid filename in save parameter
 40 - unable to open file for save
 44 - error accessing index
 48 - error scanning index
 52 - not enough memory to sort the file names
 56 - not enough memory to send output to the REXX stack
900 - memory work area is too small
```

The following example displays a list of a users files.

```
/INC REXX

   'REXLIB * Q'
   nfiles=queued()
   say 'You have the following files'
   do i = 1 to nfiles
      pull filename
      say filename
   end
```

# Program Chaining - REXX

When REXX is first invoked, it establishes itself as an *always* program. This means that it will always be given back control on the termination of any program or command it schedules. It is this technique of allowing program and command chaining that makes REXX so powerful. However, this chain can be broken by a /CAN ALL command or if one of the scheduled programs replaces REXX as the *always* program. The REXX *always* program is contained in the file called REXX. By default there is a public file on the system called REXX, which contains the default control statements defining the region size an time limits. If these defaults are not sufficient, they can be overridden by creating a private file called REXX with the appropriate parameters on the /SYS statement. Note that the /SYS statement for the initial REXX program is NOT carried over between scheduled commands. In addition since all REXX programs that are chained together should use the SAME region size, it is recommended that anyone writing a REXX program for general use, start the program with /INC REXX (not /LOAD REXX), so that the /SYS statement from the user's or system REXX file is used throughout.

# Interface with the PANEL subsystem - REXX

An interface to the PANEL facility is available through a special PANEL command in REXX. The screens are created as usual via the PANEL facility and stored as object files in the Save Library. 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.

For more information on this interface, refer to *Chapter 10. Utilities* under PANEL.

# Callable MUSIC System Functions - REXX

The following MUSIC system subroutines are callable from REXX. Each parameter should be quoted, otherwise unexpected results may occur. For example,

```
CALL TSUSER '2','USERID'
```

returns the user id in variable 'USERID'.

| | | |
|---|---|---|
| NXTCMD | CLRIN | GETRET |
| TSTIME | NOECHO | EOJ |
| TSUSER | ECHOIN | PARM |
| FILMSG | NOSHOW | |
| DELAY | NOTRIN | |
| KEEPIN | TRIN | |

# Editor Macro Facility - REXX

Editor macros can be written in the REXX language. The macros can issue Editor commands, extract information from the Editor using the EXTRACT command, call other macros, and use the MUSIO and PANEL commands. The macros cannot issue MUSIC commands.

For more information on this facility, refer the topic "Editor Macro Facility in REXX" in *Chapter 7. Using the Editor*.

# Sample REXX programs

**Example 1:**

This program prompts the user for commands and executes them.

```
/INC REXX
/* ask for the user's name */
say " Welcome to REXX, what's your name?"
pull name
say ' ' name ', when asked to enter a command, type in a MUSIC'
say " command of your choice and I'll see it gets done. If you"
say ' have trouble with the syntax of a particular command just'
say ' enter "HELP cmd" where cmd is the command name.'

  /* main prompt loop */

   do forever
     say ' Go ahead' name
     pull cmd   /*get command*/
     cmd        /*execute it */
   end
```

**Example 2:**

Calculate factorial. Note the recursive nature of the REXX language.

```
/INC REXX
/* calculate factorial */
parse arg num rest
if datatype(num,'w') •= 1 then do
    say 'I can only calculate the factorial of whole numbers.'
    exit
end
say 'The factorial of' num 'is' fact(num)
exit

fact: procedure
   arg num
   if num = 1 then return 1
   temp = fact(num-1)
   return num * temp
```

# RPG II - RPG

OS RPG II (Program Product 5740-RG1) is an enhanced version of RPG, providing users with an efficient technique for developing programs to:

1. obtain data records from input files
2. perform calculations
3. write reports
4. use table look up
5. maintain files
6. generate complete RPG II source programs from simplified specifications, and standard RPG II source specifications
7. sort input records

## Control Statements - RPG

A complete list of the RPG II and the Auto Report control statements is available in the two guides listed below.

## Usage - RPG

The following is a description of the /LOAD statements for RPG II on MUSIC:

| Statement | Description |
|---|---|
| /LOAD RPGAUTO | Autoreport execution and resulting RPG II program compilation and execution. |
| /LOAD RPG | RPG II source program compilation and execution. |
| /LOAD RPGLG | Execution of an RPG II program from object modules produced by the RPG II compiler. |
| /LOAD XMRPG | Execution of an RPG II program from load modules produced by the MUSIC linkage editor. (Similar to /LOAD XMON.) |

**Example:**

```
/FILE ... (if required)
/LOAD RPGAUTO
/JOB ...  (loader options, if required)
/OPT ...  (compiler options, if required)
.
.         (Autoreport source program)
.
/DATA
.
.         (data, if required)
.
```

The following options are available for the /OPT statement in RPGAUTO or RPG jobs:

| Option | Description |
|---|---|
| LIST | Produces an Autoreport and/or compiler listing on SYSPRINT (default). |
| NOLIST | Suppresses the list option. |
| DECK | Produces an object module from the compiled program on SYSLIN (default). |
| NODECK | Suppresses the DECK option. |

The following is a list of DDNAMES used by the various steps. These names are automatically provided by MUSIC, so a /FILE statement is not required for them. However, an overriding /FILE statement may be used if desired (for example, to direct SYSLIN output to a file).

| Step | ddname | Description |
|---|---|---|
| RPGAUTO | SYSIN | Source program input. |
| | SYSPRINT | Program listing output. |
| | SYSPUNCH | Output for the RPG II compiler. |
| | SYSLIN | Output object code if SORT/SELECT is specified. |
| | SYSLIB | Copy library containing the source code referenced by the Autoreport /COPY statement. |
| RPG | SYSIN | Source program input. |
| | SYSPRINT | Program listing output. |
| | SYSLIN | Object code output. |
| | SYSUT1 | Compiler work file. |
| | SYSUT2 | Compiler work file. |

Other DDNAMES used by the program must be defined by separate /FILE statements, placed before any /LOAD statement.

If RPG printed output (ie. output to a PRINTER device) is to be directed to a MUSIC file, a /FILE statement must be included in the job specifying a record length of 133 (LRECL(133)) and a record format of either fixed or fixed compressed (RECFM(F) or RECFM(FC)). For example:

```
/FILE REPORT NAME(REPORT.FILE) LRECL(133) RECFM(F) ...
```

Keyed or indexed files, using VSAM, are supported. Direct access by the chain and record-address file techniques, is not supported on MUSIC.

## The /COPY Statement of Autoreport - RPG

The '/COPY' statement of Autoreport is supported on MUSIC by the use of a PDS (partitioned data set), whereby members of this PDS are actually individual files in the user's library or listed in the common file index. A /FILE statement with the DDNAME SYSLIB specifying PDS name '*.RPGCOPY' is automatically supplied by MUSIC. In this way, any name specified by an Autoreport /COPY statement will have the suffix '.RPGCOPY' added to it and the system will search the appropriate Save Library for the resulting file. For example, with the /COPY statement:

```
/COPY R,ABC
```

the MUSIC file ABC.RPGCOPY will be copied into the Autoreport program.

## Sample Programs - RPG

The following two RPG II programs are provided with the system. They are:

RPG.TEST1 - RPG II program using Autoreport.

RPG.TEST2 - RPG II program.

## Reference Manuals - RPG

*DOS/VS RPG II Auto Report*, IBM Form No. SC33-6034.

*OS/VS RPG II Addendum to DOS/VS RPG II*, Auto Report, IBM Form No. SC33-6128.

*DOS/VS RPG II Language*, IBM Form No. SC33-6031.

*OS/VS RPG II Addendum to DOS/VS RPG II Language*, IBM Form No. SC33-6129.

*DOS/VS RPG II Messages*, IBM Form No. SC33-6033.

*OS/VS RPG II Addendum to DOS/VS RPG II Messages*, IBM No. SC33-6130.

# SAA RPG/370

SAA RPG/370 Program Product 5688-127, implements the high-level programming language defined by the SAA RPG Common Programming Interface (CPI). SAA RPG/370 may be used for writing a full range of batch and interactive programs that:

1. obtain data records from input files
2. perform calculations
3. write reports
4. use table look up
5. maintain files
6. generate complete SAA RPG/370 source programs from simplified specifications, and standard SAA RPG/370 source specifications
7. sort input records

## Control Statements - RPG370

A complete list of the SAA RPG/370 control statements is available in the two guides listed below.

## Usage - RPG370

The following is a description of the /LOAD statements for SAA RPG/370 on MUSIC:

/LOAD RPG370 SAA RPG/370 source program compilation and execution.

/LOAD XMRPG370 Execution of an SAA RPG/370 program from load modules produced by the MUSIC linkage editor. (Similar to /LOAD XMON.)

## Example - RPG370

```
/FILE ... (if required)
/LOAD RPG370
/JOB ...  (loader options, if required)
/OPT ...  (compiler options, if required)
.
.         (RPG/370 source program)
.
```

The following options are available for the /OPT statement:

SOURCE Produces a compiler listing on SYSCPRT. (Default on batch).

NOSOURCE Suppresses the compiler listing option. (Default on a workstation).

TEXT Produces an object module from the compiled program on SYSLIN (default).

NOTEXT Suppresses the TEXT option.

LIST Produce an pseudo assembler listing of the program.

NOLIST No pseudo assembler listing created (default).

SAAFLAG Specifies whether the compiler flags specifications not supported by SAA RPG.

NOSAAFLAG No SAA RPG specification checking is performed (default).

XREF A cross-reference list for the source program is produced.

NOXREF No cross-reference is produced (default).

The following is a list of DDNAMES used by the various steps. These names are automatically provided by MUSIC, so a /FILE statement is not required for them. However, an overriding /FILE statement may be used if desired (for example, to direct SYSLIN output to a file).

| Step | DDNAME | LRECL | Description |
|---|---|---|---|
| RPG370 | SYSIN | 80 | Source program input. |
| | SYSCPRT | 133 | Program listing output. |
| | SYSLIN | 80 | Object code output. |
| | SYSUT1 | 80 | Compiler work file. |
| | SYSUT4 | 80 | Compiler work file. |
| | SYSUT7 | 3200 | Compiler work file. |
| | SYSUT8 | 3200 | Compiler work file. |
| | SYSUT9 | 80 | Compiler work file. |
| | SYSIT | 4096 | Compiler work file. |
| | SYSTERM | 133 | Compiler error messages |
| | SYSPRINT | 133 | Compiler work file. |
| | DUMPRPG | 133 | Formatted RPG dump file |
| | SYSMSG | 150 | Compiler message files |
| | SYSMSG1 | 3200 | Compiler/Runtime message file |
| | SYSMSG2 | 3200 | Compiler/Runtime message file |

Other DDNAMES used by the program must be defined by separate /FILE statements, placed before any /LOAD statement.

When SAA RPG output to a PRINTER device, the following options MUST be specified on the /FILE statement. The options are:

```
OSRECFM(FBA) OSLRECL(xxx)  OSBLK(xxx)
```

where xxx is the length of the output line, normally 132.

For example, to have the output display at the workstation specify:

```
/FILE REPORT PRT OSRECFM(FBA) OSLRECL(132)
```

To direct the output to a file, specify:

```
/FILE REPORT NAME(REPORT.OUT) NEW(REPL) LR(132)
/ETC   OSRECFM(FBA) OSLRECL(132)  OSBLK(132)
```

Keyed or indexed files, using VSAM, are supported.

# References - RPG370

*SAA RPG/370 Programming Guide*, Form No. SC09-1335. *SAA RPG Reference Summary*, Form No. SX09-1164. *SAA RPG/370 General Information*, Form No. GC09-1336. *SAA RPG Reference*, Form No. SC09-1286.

MUSIC/SP User's Reference Guide

Part 5 - Chapter 9 (UR_P5.PS)

# Chapter 9. System Subroutines

# Chapter 9. System Subroutines

## Overview

This chapter describes the MUSIC system subroutines. Many other subroutines are also available to the MUSIC user such as the mathematical functions SQRT, and so forth.

First, a listing of the routines in functional groups will be presented. Next, an alphabetical arrangement of all these subroutines with detailed usage descriptions of each is presented.

## Functional Summary of Subroutines

### Scheduling Actions and Controlling Jobs

This group of subroutines cause a MUSIC activity to be scheduled after the current job ends.

| | |
|---|---|
| NONCAN | Make a program noncancelable. |
| CANCAN | Remove the effect of a call to NONCAN. |
| NXTCMD | Schedule a command to be run next using multi-tasking or program chaining. |
| NXTPGM | Schedule a program to be run next using program chaining. |
| PARM | Pick up the information from a /PARM control statement. |
| SAVREQ | Schedule a /SV command next. |
| SIGNOF | Schedule a /OFF next. |
| NSGNOF | Removes the request to schedule a /OFF command next. |
| SYSMSG | Suppress system information messages until after the scheduled activity is done. |
| EOJ | Terminate the current job and sets a return code. |
| GETRET | Get return code. |

### Time, Date and User Information

These routines can obtain the time and date in various formats. Information about the userid running the job and the workstation type is also available. Routines are provided to give elapsed job time. A delay routine can be used to re-activate a job after a given amount of time.

| | |
|---|---|
| CMDRET | Retrieves command strings. |
| CMDSTO | Stores command strings. |
| CODON | Determines if a userid is signed on to MUSIC. |
| DATCON | Converts date data from one format to another. |
| DATCN2 | Similar to DATCON with a more thorough analysis of the argument list. |
| DELAY | Specify time delay before job is re-activated. |
| GETID | Gets ownership id from a file name or data set name. |
| STATUS | Returns the time of day, current date, service units used, etc. |
| TIMCON | Converts time data from one format to another. |
| TIMDAT | Gets time of day (by TSTIME) and date (by TSDATE). |
| TIMOFF | Job time is displayed. |
| TIMON | Reset time counter used in TIMOFF. |
| TSDATE | Give current date. |
| TSTIME | Give current time of day or execution time. |
| TSUSER | Provide userid and workstation type. |

## Dynamic Access

These routines allow you to read and write files from FORTRAN without having to pre-define the file names on /FILE statements. The files are opened and closed dynamically.

| | |
|---|---|
| OPNFIL | Open a file. |
| CLSFIL | Close a file. |
| SETINF | Specify information for a new file. |
| FILMSG | Get error description text. |
| PURGE | Delete a file. |
| FRSTOR | Frees main storage for use via an array. |
| GTSTOR | Gets main storage for use via an array. |
| QFOPEN | Opens a file and defines buffer area |
| QFCLOS | Closes a file and releases buffer |
| QFREAD | Reads next logical record |
| QFBKRD | Reads previous logical record |
| QFRBA | Sets RBA for next read |
| QFREW | Rewinds the file |

## SYSIN Read Routines

These routines allow you to dynamically specify which file to read from the Save Library. They also allow you to inspect where the SYSIN (unit 5) data is coming from and issue messages.

| | |
|---|---|
| SYSINR | Dynamically open a file for reading. |
| SYSINM | Give message identifying file and line number. |
| SYSINE | Inspect error return codes for sysin. |
| SYSINL | Inspect SYSIN nesting stack. |

## Block Transfer

These routines allow reading and writing of large blocks to disk files, bypassing the logical blocking and buffering operations. A large number of separate logical files are allowed.

| | |
|---|---|
| GULPDF | Defines file characteristics for GULP routines. |
| GULPRD | Read in GULP mode. |
| GULPWR | Write in GULP mode. |

## FORTRAN I/O

These routines are used in conjunction with FORTRAN I/O. They can re-read the last logical record, do I/O to an in-core buffer, allow mixed free-and fixed-format I/O, and close a direct access file.

| | |
|---|---|
| MCNCAT | Replace VS Fortran's CNCAT# concatenation routine. |
| REREAD | Re-read last logical record. |
| CORE | DO I/O to buffer in main storage. |
| XGCON | Allow mixed free and formatted I/O. |
| XGCOFF | Turn off XGCON mode. |
| CLOSDA | Close direct access buffers. |
| FRMTDA | Erase the contents of a direct-access data set. |
| BIGBUF | Allow record lengths of greater than 133 bytes to be used. |

## ASCII Terminals

| | |
|---|---|
| TPOPEN | Turn off line folding at col 72 on a TTY terminal. |
| TPCLSE | Turn on line folding at col 72 for TTY terminal. |
| NOCRLF | Stop automatic return at the end of lines. |
| CRLF | Re-instate automatic return at end of line. |
| NOTRIN | Suppress translation of terminal input. |
| TRIN | Cancel a call to NOTRIN and resume the normal translation of terminal input. |

## 3270-Type Workstations

These routines allow you to control the display of the input area on 3270-type workstations when performing a conversational read.

| | |
|---|---|
| ADTOXY | Convert a 3270 2-byte screen address to row and column. |
| KEEPIN | Do not erase the contents of the input area. |
| CLRIN | Erase the contents of the input area. |
| NOECHO | Do not display the contents of the input area on the output area. |
| ECHOIN | Display the contents of the input area on the output area. |
| NOSHOW | Do not display the characters being entered to the input area. |
| SHOWIN | Display the characters being entered to the input area. |

## Workstation Prompting

These routines allow you to remove and restore the conversational read prompt.

| | |
|---|---|
| NPRMPT | Remove prompting entirely |
| PROMPT | Do prompting again |

## Workstation Features

These routines can be used to set TABS, turn on and off line folding, nonstop mode, and type element return functions.

| | |
|---|---|
| TABS | Informs MUSIC of desired TAB settings. |
| NOPAUS | Set nonstop mode. Used mainly for 1050 terminals. |
| PAUSE | Turn off non-stop mode. Used mainly for 1050 terminals. |
| TEXTLC | Lower case letters input during execution are to be preserved as such. |
| TEXTUC | Lower case letters input during execution are to be converted to their upper case equivalents. |

## Controlling Output

| | |
|---|---|
| STOPSK | Stop /SKIP operation requested by the user. |

## Controlling Tape

| | |
|---|---|
| BACKSP | Do a backspace operation on a tape file defined by a ddname. |

## Manipulating Bits

Routines that work on the 8 bits of a byte. Each bit of the byte has the two possible values of ON or OFF.

| | |
|---|---|
| BITON | Turn bit on. |
| BITOFF | Turn bit off. |
| BITFLP | Reverse a bit. |
| TBIT | Test if bit is on or off. |

## Character Translation

| | |
|---|---|
| CTRAN | Translate each occurrence of one character to another. |
| FIXSCR | Translates hex characters to blanks. |
| TOLC | Translates a string of characters to lower case. |
| TOUC | Translates a string of characters to upper case. |
| TRANSL | Modify characters using a translate table. |

## Moving Bytes and Words

These routines get or move storage on a byte or word basis. One routine can set many bytes to zero.

| | |
|---|---|
| BYTE | Get first byte of arg. |
| FILL | Fill storage with a specified character. |
| HWORD | Get first 2 bytes of arg. |
| LMOVE | Move bytes from one place to another. |
| LBMOVE | Same as LMOVE but on any byte boundary. |
| MOVE | Move words in storage. |
| ZERO | Set storage to zeros. |

## Converting Character Strings

| | |
|---|---|
| C2D | Convert a numeric character string to a decimal value (double precision). |
| C2I | Convert a numeric character string to an integer value. |
| C2R | Convert numeric character string to a decimal value (single precision). |
| C2X | Convert a character string to hexadecimal. |
| DECN | Convert a numeric string to an integer value. |
| DECOUT | Convert an integer value to a character string, similar to Fortran I format. |
| I2C | Convert an integer value to a character string. |
| I2X | Convert an integer to a hexadecimal string. |
| X2C | Convert a hexadecimal string to characters. |
| X2I | Convert a hexadecimal string to an integer value. |
| MA2E | Convert from Ascii to EBCDIC (1-to-1 mapping for all 256 characters). Same calling sequence as A2E. |
| ME2A | Convert from EBCDIC to Ascii (also 1-to-1). Same calling sequence as E2A. |
| TBAS64 | Encode data using Base64 method (MIME standard). |
| FBAS64 | Decode Base64 data. |
| B64TXT | Processes Base64 data: decode, separate into records by CRLF, and convert from Ascii to EBCDIC. Record input and output is done by calling user-supplied subroutines. This routine could be used by MAIL to display MIME text. |
| UUENC | Encode data using the UUENCODE method. Similar to Base64, but uses different character table and does not remember the exact original length. |
| UUDEC | Decode UUENCODE'd data. |

## Manipulating Strings

| | |
|---|---|
| ABBREV | Test whether a string is a true abbreviation for a keyword. |
| CENTER | Center a string. |
| GETOP | Get the parameters from a character string. |
| LJUST | Left justify a string. |
| LN | Gets the length of a character string. |
| NXWORD | Get the next word from a string. |
| PROCOP | Process an option item in the format abc( xyz ). |
| RJUST | Right justify a string. |
| RVRS | Reverse the order of characters in a string. |
| SEP | Separate out fields separated by 1 or more blanks/commas. |
| WORD | Separate a string into words. |

## Character Searching

| | |
|---|---|
| FNDALL | Find all specified *reference* characters in a string. |
| FNDCHR | Find the first *reference* character in a string. |
| LOCATE | This routine finds the first occurrence of one string within another string. |
| VERALL | Verify all characters of a string are within a set of reference characters and returns all mismatches. |
| VERIFY | Verify all characters of a string are within a set of reference characters and returns the first mismatch. |

## Comparing Bytes and Words

Routines to compare numbers or character strings for equality (and so forth).

| | |
|---|---|
| EQUAL | Compare strings for equality. |
| EQUALB | Compare byte strings for equality. |
| LCOMP | Compare strings. |
| LBCOMP | Compare byte strings for equality. |

## Logical Operations

Perform logical operations and shift bits within words.

| | |
|---|---|
| LAND | Logical AND. |
| LOR | Logical OR. |
| LXOR | Logical exclusive OR. |
| LCOMPL | Logical complement. |
| LSHFTL | Do left shift operation. |
| LSHFTR | Do right shift operation. |

## Sorting

See "Sorting Routines" in *Chapter 10. Utilities* for larger disk sort operations.)

| | |
|---|---|
| DSORT | Generalized disk sort subroutine, see *Chapter 10. Utilities*. |
| SSORT | Sort arrays in main storage. |
| SRTMUS | Similar to DSORT, see *Chapter 10. Utilities* |

## Generating Random Numbers

This routine, with its several function calls, allows for various types of random number generation.

| | |
|---|---|
| RSTART | Random number generation |

## PANEL Support

| | |
|---|---|
| MODFLD | Redefine the attributes of modifiable fields by field number of screens designed using PANEL. |
| MODOPT | Modify panel options previously defined using the PANEL subsystem. |
| PANFLD | Query information about PANEL fields. |

## Debugging

| | |
|---|---|
| DEBUG | Invokes the Debug Facility at various points in your program. |

## TCP/IP Sockets

| | |
|---|---|
| CARGCALL | Guarantees that arguments are passed correctly for C programming. |

## Index Text Searching (ITS)

| | |
|---|---|
| ITSFID | Initialization/dynamic call |
| ITSFIW | Initialization/workarea call |
| ITSFOP | Open a search set |
| ITSFSS | Search string |
| ITSFOR | Order result list |
| ITSFRE | Retrieve results |
| ITSFCL | Close search |

## Miscellaneous

| | |
|---|---|
| LOCNAM | Change an output file name to make it local to the current directory (if necessary). |

# Subroutines Listed Alphabetically

## ABBREV

This subroutine tests whether a string is a true abbreviation for a keyword.

**Calling Sequence:** k=ABBREV(a,len,kw,minl,maxl)

**Arguments:**

a is a true abbreviation.

len is the length in bytes.

kw is the keyword.

minl is the minimum length of the abbreviation.

maxl is the maximum length of the abbreviation.

ABBREV returns k=1 if *a* (of length *len* bytes) is a true abbreviation for *kw* (minimum abbreviation length *minl,* total length *maxl),* otherwise k=0. Should have 1 <= minl <= maxl <= 256.

```
        return k=1 if:       len>0
                       and   len>=minl
                       and   len<=maxl
                       and   equal(a,kw,len)
```

See also: NXWORD, PROCOP, and GETOP.

## ADTOXY

This subroutines converts a 3270 2-byte screen address to row and column numbers.

**Calling Sequence:** CALL ADTOXY(addr,width,row,column).

**Arguments:**

addr (input) 2-byte screen addr, either 12-bit coded form or 14-bit uncoded form.

width (input) screen width (e.g. 80 or 132).

row (output) row number (>=1).

column (output) column number (1 to width).

## BACKSP

This routine preforms a backspace operation on a tape file defined by a ddname. It can be used by a COBOL or PL/I program to add data to the end of an existing tape file. Refer to the discussion on /FILE for tape.

**Calling Sequence:** CALL BACKSP('ddname ')

# BIGBUF

(Fortran G1 only) This subroutine is used to increase the size of the maximum record length handled by FORTRAN sequential I/Os. If this subroutine is not called, the maximum is 133 bytes.

To allow record lengths of more than 133, call BIGBUF and supply an array of length *n* bytes. The system uses a work area buffer for all input/output except direct access. BIGBUF is normally called once at the beginning of the job, before any input/output is done.

**Calling Sequence:** CALL BIGBUF(buffer,n)

**Arguments:**

buffer — is the name of the array that is used as an I/O buffer. Since the array must start on a fullword boundary, it is convenient to use an array of type INTEGER*4 or REAL*4.

n — is any number 133 or more indicating the record length. Regardless of the limit defined by BIGBUF, the maximum record length in a file may be subject to other limitations. For further details see *Chapter 4. File System and I/O Interface* of this guide.

**Example:**

The following FORTRAN program writes three 400-byte records to a temporary file, then reads the records back and displays them. If BIGBUF were not called, this program would give the error message: IHN212I, END OF RECORD ON UNIT 1.

```
/FILE 1 NAME(&&TEMP) NEW DELETE LRECL(400)
      INTEGER BUF(100),A(80)
      CALL BIGBUF(BUF,400)
      WRITE(1,10) (I,I=1,240)
10    FORMAT(80I5)
      REWIND 1
15    READ(1,10,END=20) A
      WRITE(6,*) A
      GO TO 15
20    STOP
      END
```

# BITFLP

This subroutine is identical to BITON except that the referenced bit is *reversed*. That is, a 0 is set to 1, and a 1 is set to 0. This routine is an INTEGER function and must be declared as such in your program.

**Calling Sequence:** n=BITFLP(a,k)

**Arguments:**

a — location of the byte.

k — is a number between 0 and 31 indicating the bit position.

# BITOFF

This subroutine is identical to BITON except that the referenced bit is set to 0 instead of 1. This routine is an INTEGER function and must be declared as such in your program.

**Calling Sequence:** n=BITOFF(a,k)

**Arguments:**

a location of the byte.

k is a number between 0 and 31 indicating the bit position.

# BITON

This FORTRAN INTEGER function is used in conjunction with TBIT, BITOFF, and BITFLP. It returns the current value of bit *k* (k=0,1,2,...) of location *a*, and sets that bit to 1. This routine is an INTEGER function and must be declared as such in your program.

**Calling Sequence:** n=BITON(a,k)

**Arguments:**

a location of the byte.

k is a number between 0 and 31 indicating the bit position. For example, if *k* is 24, the value of the first bit of the fourth byte of *a* is returned and that bit is set to 1. The first bit position is referred to as bit 0. If *k* is negative, a value of zero is returned and the bit is not changed.

# BYTE

This subroutine is a FORTRAN INTEGER function which returns the value (0 to 255) of the first byte of the argument *x*. This routine is an INTEGER function and must be declared as such in your program.

**Calling Sequence:** n=BYTE(x)

# B64TXT

This routine processes BASE64 data, as in mime mail data, decodes it, separates it into records, and converts it from ASCII to EBCDIC. Records are assumed to be separated by ASCII CRLF (X'0D0A'). Input records (max 80 bytes each) are read by calling user-supplied routine B64TRD. Output records are written by calling user-supplied routine B64TWR. (this routine is reentrant.)

In the input BASE64 data, blanks and line breaks are ignored. Also, control characters (less than x'40') are also ignored. An input quartet (group of 4 encoded bytes) may be split by these characters or line breaks. If invalid characters are found in the BASE64 data, the corresponding output triples are replaced by "???".

Routines used: FBAS64 - decode base64 data; MA2E translate from 8-bit ascii to ebcdic.

**Calling Sequence:** CALL B64TXT(work,wrklen,mxolen,argrd,argwr,numout,retcod)

**Arguments:**

work — a work area, on a doubleword boundary.

wrklen — (input) length of the work area. Should be at least 400 + mxolen.

mxolen — (input) maximum length of output records. when looking for crlf's to separate the records, records longer than this are truncated. if mxolen<1, vaule 1 is used.

argrd — (input) this argument is passed to B64TRD routine.

argwr — (input) this argument is passed to B64TWR routine.

numout — (output) number of output records.

retcod — (output) return code:
0 normal, no errors.
1 work area is too small.
2 terminated by error from b64trd routine.
3 terminated by error from b64twr routine.
4 1 or more bad characters were found in the base64 data. the output text may be incorrect.
5 1 or more output records were truncated.
6 both retcod conditions 4 and 5.

The following is the calling sequences for user-supplied record read/write routines:

**Calling Sequence:** CALL B64TRD(argrd,rec,maxlen,len,k)

**Arguments:**

argrd — same as arg passed to B64TXT.
rec — buffer to receive input record.
maxlen — max length of input record (= buffer size). currently maxlen is 80.
len — (output) actual length of record (0 to maxlen). B64TXT skips 0-length input records.
k — (output) 0=normal, 1=eof, 2=error (B64TXT stops). Note: if an EOF quartet (one with the special pad character "=") is found in the input, B64TXT stops before eof is received from b64trd.

**Calling Sequence:** CALL B64TWR(argwr,rec,len,k)

**Arguments:**

argwr — same as arg passed to b64txt.
rec — record to be put out.
len — length of the record (0 or more).
k — (output) 0=normal, 1=error (b64txt stops).

# CANCAN

A call to this routine removes the effect of a previous call to the NONCAN routine. It makes the program cancelable by the /CANCEL command. A call to CANCAN does not take effect until after the next

conversational read or call to CLSOUT.

**Calling Sequence:** CALL CANCAN

# CARGCALL

This subroutine is intended for C programmers calling TCP/IP socket routines. When called before the first socket routine, it guarantees that the arguments are passed in the manner C requires.

**Calling Sequence:** CALL CARGCALL();

# CENTER

This routine centers a substring, within a 1-to 256-character string, defined as the first non blank character from the left and the last non blank character. The substring is then placed at the center of the string.

**Calling Sequence:** CALL CENTER(string,strlen,pad)

**Arguments:**

| | |
|---|---|
| string | is a 1-to 256-character string to be centered. |
| strlen | is the length of the string, an integer in the range of 1-256. When strlen=0 no action is taken by CENTER. |
| pad | all leading and trailing blanks in the string are converted to the pad character, after the text has been centered. |

**Example:**

Given that:

```
STRING --> 'bb12345bbbbb'
STRLEN --> 12
PAD    --> 'b'

CALL CENTER(STRING,STRLEN,PAD)
```

Returns:

```
STRING --> 'bbb12345bbbb'
STRLEN --> 8
```

# CLOSDA

(Fortran G1 only) A call to this subroutine *closes* a direct access file specified by *unit.* This close operation causes the contents of buffers to be written out on disk. Also, a direct access read following a call to CLOSDA actually reads from disk, rather than from a buffer in main storage. This is important for applications where several programs share a direct-access file. You may still use the file for further direct-access operations. Note that direct-access data sets may not be closed by the system unless the job terminates

normally.

**Calling Sequence:** CALL CLOSDA(unit)

# CLRIN

A call to this subroutine cancels the effect of a previous call to the KEEPIN subroutine.

**Calling Sequence:** CALL CLRIN

# CLSFIL

Dynamically close a file. Refer to the section "Dynamic Access to Files" for a description of this routine.

# CMDRET

This routine retrieves command strings from a circular buffer of previously stored commands. This works in the same way as the store/retrieve (F12) when in *Go. The CMDSTO is used to store the commands strings in the buffer.

**Calling Sequence:** CALL CMDRET(cmd,cmdlen,maxlen,cmdbuf)

**Arguments:**

| | |
|---|---|
| cmd | a 1-126 character string returned from the command buffer |
| cmdlen | the length of the string CMD is returned. |
| maxlen | the maximum length of the string CMD (1-126) |
| cmdbuf | a 256 character string to be used as the command buffer. |

*Notes:*

1. Only the first 128 bytes of CMDBUF are used, the other 128 bytes are used as a temporary buffer.
2. Successive entries of the same CMD string will not be stored.
3. This routine is re-entrant.

# CMDSTO

This routine stores a command string into a circular buffer for later retrieval. This works in the same way as the store/retrieve (F12) when in *Go. The CMDRET is used to retrieve from the commands from the buffer, in the order they were entered.

**Calling Sequence:** CALL CMDSTO(cmd,cmdlen,maxlen,cmdbuf)

**Arguments:**

| | |
|---|---|
| cmd | a 1-126 character string to be placed in the command buffer. |
| cmdlen | the length of the string CMD (1-126). |
| maxlen | the maximum length of the string CMD (1-126). |
| cmdbuf | a 256 character string to be used as the command buffer. |

*Notes:*

1. Only the first 128 bytes of the CMDBUF are used, the other 128 bytes are used as a temporary buffer.
2. Successive entries of the same CMD string will not be stored.
3. This routine is re-entrant.

# CODON

This routine reads the terminal control blocks (TCB) and determines if the code passed is signed on to MUSIC and returns the TCB number and the code's terminal mode.

**Calling Sequence:** CALL CODON(code,tcbno,mode,irc)

**Arguments:**

| | |
|---|---|
| code | seven-character user id |
| tcbno | return terminal (TCB number) 2 character integer |
| mode | return terminal mode<br>0= not active<br>1= *go<br>2= break<br>3= conversational<br>4= spooled input<br>5= running a job<br>6= FSIO<br>7= command execution from a prog |
| irc | Set to 0 if code is not active, set to 1 if code is active. |

# CORE

This subroutine for FORTRAN allows reading and writing from a buffer in memory, without physical I/O.

Executing this call statement causes the next formatted or list-directed READ or WRITE to transmit from or to the data area in core beginning at location buffer, and extending length bytes.

**Calling Sequence:** CALL CORE(buffer,length)

**Arguments:**

buffer is an array name.

length is the length of the array in bytes. Length must be at least 4 and up to 32000.

*Notes:*

1. The call to CORE must be executed before each formatted or list-directed READ or WRITE (that is to use an incore buffer) is executed. The unit number in the READ or WRITE statement is ignored. CORE must not be called twice without an intervening formatted or list-directed READ or WRITE statement. The I/O request must not attempt to read or write more than one logical record.

2. Declaring the buffer to be of type LOGICAL *1 has the advantage that each character in the buffer can be accessed by subscripting. The buffer is filled with blanks before any data is written into it.

3. Unformatted READ or WRITE statements, and BACKSPACE or REWIND statements must not be executed between the CALL CORE and the READ or WRITE statement.

4. If REREAD and CORE are used within the same program, REREAD is temporarily disabled when CORE is called.

   For example:

```
      LOGICAL *1 BUFF(80)
    5 CALL REREAD
   10 READ(5,100)X
   20 CALL CORE(BUFF,80)
   30 WRITE(N,102)X
   40 CALL CORE(BUFF,80)
   50 READ(99,103)Y
   60 READ(99,104)Z
```

   *x* is written into the incore buffer BUFF by statement 30. Statement 50 reads from the incore buffer into *y*. Statement 60 rereads the record read by statement 10.

**Example:**

```
*Go
list sample
*In progress
/LOAD VSFORT
      DIMENSION A(5)
      LOGICAL *1 BUFF(80),NBUFF(80)
      LOGICAL EQUAL,B1
      DO 1 I=1,5
    1 A(I)=10**I
      WRITE(6,2)A
    2 FORMAT('0',5F12.0)
      CALL CORE(BUFF,80)
      WRITE(N,8)A
    8 FORMAT(5F12.0)
C     NOW COMPRESS MULTIPLE BLANKS IN
C     BUFF TO ONE BLANK, BY COPYING TO NBUFF
      NEW=1
      B1=.FALSE.
```

```
      DO 3 K=1,80
      IF(EQUAL(BUFF(K),' ',1)) GO TO 4
      B1=.FALSE.
    5 NBUFF(NEW)=BUFF(K)
      NEW=NEW+1
      GO TO 3
    4 IF (B1) GO TO 3
      B1=.TRUE.
      GO TO 5
    3 CONTINUE
      NEW=NEW-1
      WRITE(6,7)(NBUFF(I),I=1,NEW)
    7 FORMAT('0',80A1)
      CALL EXIT
      END
*End
*Go
sample
*In progress
MAIN   = 0003DC
003AA8 BYTES USED
EXECUTION BEGINS

         10.        100.       1000.      10000.     100000.

10. 100. 1000. 10000. 100000.
*End
*Go
```

# CRLF

A call to this subroutine cancels the effect of a call to the NOCRLF subroutine. This subroutine is for ASCII terminals only.

**Calling Sequence:** CALL CRLF

# CTRAN

This routine locates and translates every occurrence of a specified character to another character. The number of characters changed is returned.

**Calling Sequence:** CALL CTRAN(string,strlen,oldchr,newchr,kount)

**Arguments:**

string is character string to be processed of length *strlen.*

strlen is an integer specifying the length of the *string.*

oldchr is the target character in string to be converted to the character specified by *newchr.*

newchr is the new character for which each occurrence of *oldchr* will be swapped in *string.*

kount is an integer returned that reports the number of characters changed as specified.

**Example:**

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8
OLDCHR --> 'b'
NEWCHR --> '*'

CALL CTRAN(STRING,STRLEN,OLDCHR,NEWCHR,KOUNT)
```

Returns:

```
STRING --> '**12345*'
STRLEN --> 8
OLDCHR --> 'b'
NEWCHR --> '*'
KOUNT  --> 3
```

# C2D

This subroutine converts a numeric character string to its double precision decimal value. Conversion from character number to a decimal is done by separating the whole and the fractional part. These are then converted to their integer values. *j* and *k* are first converted to long-format floating point. Then *x* is computed from *j.k*10**n.* where *j* is the whole, *k* is the fractional part and *n* is the exponent.

**Calling Sequence:** CALL C2D(string,strlen,x,irc,badchr,iexp)

string is a 1-to 256-character string on which a substring of character numbers is to be converted to short or long format floating point number, in other words, a real number.

strlen is the length of the string, an integer in the range of 1-256.

x is the real floating point number returned.
*x* must be declared as REAL*8 (double precision).

irc is the return code describing the conversion.
irc=-1 The string was blank.
irc=0 Conversion was successful.
irc=1 An illegal character was found in the string.
irc=2 Exponent overflow (IEXP>75).
irc=3 Exponent underflow (IEXP<-75).

*Note:* *x* is set to zero when *irc* is not equal to 0.

badchr A one byte variable where the bad character (non integer character) is returned when *irc=1.*

iexp An integer variable where the exponent of the number being converted is returned when *irc=2* or *irc=3.*

# C2I

This routine converts the substring, within a 1-to 256-character string, defined as the first non blank character from the left and the last non blank character, to an integer.

**Calling Sequence:** CALL C2I(string,strlen,i,irc,badchr)

**Arguments:**

string — is a 1-to 256-character string on which a substring of character numbers is to be converted to integer.

strlen — is the length of the string, an integer in the range of 1 to 256.

i — is the integer returned.

irc — is the return code describing the conversion.
irc=-1 the string was blank.
irc=0 conversion was successful.
irc=1 an illegal character was found in the string.
irc=2 conversion not possible, number was to large.

*Note:* *i* is set to zero when *irc* is not equal to 0.

badchr — a one byte variable where the bad character (non-integer character) is returned when *irc=1*.

**Example:**

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8

CALL C2I(STRING,STRLEN,I,IRC,BADCHR)
```

Returns:

```
STRING --> 'bb12345b'
STRLEN --> 8
I      --> 12345
IRC    --> 0
BADCHR --> 'b'
```

# C2R

This subroutine converts a numeric character string to its single precision decimal value. Conversion from character number to a decimal is done by separating the whole and the fractional part. These are then converted to their integer values. *j* and *j* are first converted to long-format floating point. Then *x* is computed from *j.k*10**n.* where *j* is the whole, *k* is the fractional part and *n* is the exponent.

**Calling Sequence:** CALL C2R(string,strlen,x,irc,badchr,iexp)

**Arguments:**

| | |
|---|---|
| string | is a 1 to 256 character string on which a substring of character numbers is to be converted to short or long format floating point number. i.e. a real number. |
| strlen | is the length of the string, an integer in the range of 1-256. |
| x | is the real floating point number returned.<br>*x* must be declared as REAL*4 (single precision). |
| irc | is the return code describing the conversion.<br>irc=-1 The string was blank.<br>irc=0 Conversion was successful. ..text<br>irc=1 An illegal character was found in the string.<br>irc=2 Exponent overflow (IEXP>75)<br>irc=3 Exponent underflow (IEXP<-75)<br><br>*Note:* that *x* is set to zero when *irc* is not equal to 0. |
| badchr | A one byte variable where the bad character (non-integer character),is returned when *irc=1*. |
| iexp | An integer variable where the exponent of the number being converted,is returned when *irc=2* or *irc=3*. |

# C2X

This subroutine converts (unpacks) a EBCDIC character string to its hexadecimal character representation.

**Calling Sequence:** CALL C2X(string,strlen,hexstr)

**Arguments:**

| | |
|---|---|
| string | is a string of length strlen to be converted to its hexadecimal representation. |
| strlen | is the length of the string, in the range of 1 to 256. When *strlen* =0 no conversion takes place. |
| hexstr | is the output string where EBCDIC hexdigits representing the original string are returned. The character string must be at least twice the length of string. |

**Example:**

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8

CALL C2X(STRING,STRLEN,HEXSTR)
```

Returns:

```
STRING --> 'bb12345b'
STRLEN --> 8
HEXSTR --> '4040F1F2F3F4F540'
```

# DATCN2

This subroutine is a version of DATCON that does a more thorough analysis of the argument list passed. When an error occurs it passes an error number to the calling program. It does not stop the job, nor does it print error messages. Except for the insertion of the **irc** variable in the calling sequence, DATCN2 parameters are identical to DATCON.

Enhanced error checking includes:

- Checks if too many or too few parameters are passed.

- Except for type 10, all input dates are checked for valid month, that the corresponding number of days in that month is correct, and that the number of days is not greater than a full year. the leap year is taken into account on all verifications.

- For types 4 and 6 the integer year can be expressed as 1989 or 89. "1900" is assumed.

- For type 10, a check is made that it is a packed decimal value only. Since this data was generated by datcn2, datcon, or tsdate, further checking is not required.

**Calling Sequence:** CALL DATCN2(irc,m,arg1,{arg2,arg3},n,arg1,{arg2,arg3})

**Arguments:**

irc is the one of the following return codes:

irc=0 no error detected conversion successful
irc=1 input data conversion type is invalid
irc=2 output data conversion type is invalid
irc=3 input data (date) is invalid
irc=4 wrong number of arguments detected.

See DATCON for descriptions of the other arguments.

# DATCON

This subroutine is used to convert date data from one format to another. DATCON checks incoming dates for errors and stops the job and writes an error message. In addition, no attempt is made to convert the bogus dates; blanks are returned in character fields and zero values in numeric fields.

**Calling Sequence:** CALL DATCON(m,arg1,{arg2,arg3},n,arg1,{arg2,arg3})

**Arguments:**

m is one of the numbers listed below (except 5) representing the input date format.

n is one of the numbers listed below representing the output date format.

1 16 byte date, for example: 'FRI SEP 06, 1989'
2 is an 8 byte date in the form MM/DD/YY, for example: 09/06/89.
3 is an 8 byte date, for example: 06SEP89
4 *arg1* is the integer year (1989); *arg2* is the integer date of the year (249).
5 is the integer day of the week (Sunday is 1, Monday is 2).

6 *arg1* is the integer month (5); *arg2* is the integer day (26); *arg3* is the integer year (1989).
7 is an 8 byte date in the form DD/MM/YY, for example: 26/01/89.
8 is an 8 byte date in the form YYMMDD, for example: 890126.
9 is an 8 byte date in the form DDMMYY, for example: 260189.
10 is a 4 byte date in packed decimal 0077DDDZ format, for example: 0089360C.

# DEBUG

Calls to subroutine DEBUG ($SUB:DEBUG.OBJ) can be inserted at various points in a program, to invoke the Debug Facility when execution reaches those points. If Debug is not active, the calls are ignored. The usage in a VS Fortran program is "CALL DEBUG" (no arguments). /INCLUDE $SUB:DEBUG.OBJ must be inserted in the job.

Specify the DEBUG option on /JOB (or on the member name statement after /LOAD XMON). At the beginning of the program, you are presented with the Debug screen. Press F2 to run the program. When a CALL DEBUG is reached in the program, the Debug Facility is invoked and you can do debug operations from the Debug screen. Press F2 to resume execution.

This technique is useful when it is not feasible to trace the entire program or set Debug break points. Note that VS Fortran programs are difficult to trace, because Fortran run-time library modules such as IFYVRENC are normally in the Link Pack Area, which is read-only storage.

# DECN

This subroutine converts a numeric character string to an integer value. The string is of length from 1 to 9. A valid string consists of decimal digits, without any leading blanks.

**Calling Sequence:** CALL DECN(string,len,ivalue,irc)

**Arguments:**

string is a numeric character string specified by the caller.

len is the length of the string from 1 to 9.

ivalue is the output integer value.

irc is one of the following return codes:
*irc=0* successful conversion.
*irc=1* input string is incorrect.

# DECOUT

This routine converts an integer value to a printable character string, in the same way as done by Fortran I format. The output is right justified in the area, preceded by blanks, and a minus sign ("-") is supplied if the number is negative. If the area is too short, it is filled with asterisks ("*"). This routine is re-entrant (it does not modify itself).

**Calling Sequence:** CALL DECOUT(string,len,ivalue)

**Arguments:**

string is the output character string.

len is the length of *string* set by the caller. The range is from 0 to 256.

ivalue is the input number.

# DELAY

A call to this routine suspends the execution of the job for at least *n* seconds. Use the subroutine CLSOUT before a long delay if you wish to force all generated output to be printed before the delay.

**Calling Sequence:** CALL DELAY(n)

**Arguments:**

n is the number of seconds for the delay period. You should use a delay period of at least 1 second.

# DSORT

DSORT is a generalized disk sort subroutine callable from many of MUSIC's high-level languages such as FORTRAN. For information see "Sorting Routines" in *Chapter 10. Utilities*.

# ECHOIN

A call to this subroutine cancels the effect of a previous call to the NOECHO subroutine.

**Calling Sequence:** CALL ECHOIN

# EOJ

This routine terminates the current job and sets the return code. (See also the GETRET subroutine.)

**Calling Sequence:** CALL EOJ(irc)

**Arguments:**

irc is an integer number supplied by the caller. It is put into register 15 when the job ends. A value of 0 usually means normal end, while a nonzero value usually indicates an error condition.

# EQUAL

This subroutine is a FORTRAN LOGICAL function used for comparing bytes. If the compared bytes are identical, EQUAL is set to TRUE. If they are not identical, EQUAL is set to FALSE. This function must be declared as LOGICAL in your program. (See also EQUALB.)

**Calling Sequence:** v=EQUAL(a,b,n)

**Arguments:**

a is one location of bytes.

b is another location of bytes.

n is the number of bytes to compare. If *n* is less than 1, EQUAL is set to TRUE.

**Example:**

```
LOGICAL EQUAL
REAL *8 ALPHA,BETA
:
IF(EQUAL(ALPHA,BETA,6)) GO TO 10
```

# EQUALB

This is a FORTRAN LOGICAL function used for comparing bytes. If the compared bytes are identical, EQUALB is set to TRUE. If they are not identical, EQUALB is set to FALSE. This function must be declared as LOGICAL in your program.

**Calling Sequence:** v=EQUALB(a,ka,b,kb,n)

**Arguments:**

a is one location of bytes.

ka indicates which byte to start with at location *a*. (The first byte is referred to as byte 1, not 0.)

b is another location of bytes.

kb indicates which byte to start with at location *b*. (The first byte is referred to as byte 1, not 0.)

n is the number of bytes to compare. If *n* is less than 1, EQUALB is set to TRUE.

**Example:**

```
LOGICAL EQUALB
REAL *8 ALPHA,BETA
:
IF(EQUALB(ALPHA,2,BETA,4,2)) GO TO 10
```

# FBAS64

This subroutine decodes BASE64 data. This is the "base 64" encoding scheme used for e-mail mime, as defined in RFC 1521. Base 64 encodes any 8-bit text to printable characters, producing 4 output characters for each 3 input bytes. See the translate table in this routine for the 64 printable characters used. An additional character "=" is used for padding at the end of the encoded text if the original length is not a multiple of 3. See TBAS64 routine for additional notes on the pad character and for examples of encoding. (This routine is re-entrant.)

This routine decodes the base 64 data, back to the original data. See also: TBAS64 - encode to base 64; and UUDEC - uudecode-style decode.

**Calling Sequence:** CALL FBAS64(intxt,inlen,outtxt,outlen,retcod,displ)

**Arguments:**

intxt — input data (encoded as base 64). It is assumed to start on a quartet (group of 4 bytes in the encoded text) boundary. Note: intxt is destroyed by this routine (tranlated in place), so the caller should pass a copy of the original data, if it is to be used again. intxt is destroyed only up to (not including) the first blank or pad character ("=") or invalid character or to the end of the last full input quartet.

inlen — length of input data. If 0 or less, this routine sets outlen=0, retcod=0, and displ=0 and no other action is taken.

outtxt — area to receive output (decoded) data. It must be large enough to hold the output data. Maximum possible output is ((inlen+1)/4)*3 bytes, where / is integer divide i.e. discard the remainder.

outlen — this argument sets outlen to the length of the output data. This is usually a multiple of 3, but may be otherwise if the pad character ("=") is found in the input, indicating end of data. outlen is always the number of bytes stored into outtxt.

retcod,displ — the routine sets these integer arguments to indicate various cases, errors, and ending conditions. retcod is a return code and displ is a displacement within intxt, usually indicating how many input characters have been processed. Scanning stops when a blank is found, or after an EOF quartet (with "=" pad char) is found, or an invalid character is found; this we call the "scan end".

1. if the scan ends immediately follows a quartet, and the last quartet is not an EOF quartet, retcod=0 and displ=displacement to scan end i.e. 4*(number of quartets processed). outlen=3*(number of quartets processed).
2. if scan ends immediately follows an eof quartet, displ is set as in case (1), and retcod=-1 (if quartet is of the form "xxx=") or -2 (if "xx=="). an exception is that if the second "=" is not actually present in "xx==", displ is to after the first "=", i.e. 4*(no. of quartets) - 1. outlen=exact length of original text.
3. if scan end is within a quartet (other than the exception in case (2)), retcod=number of of chars in last quartet before the scan end i.e. 1 to 3, and displ=displacement to end of the last complete quartet. outlen=3*(number of complete quartets processed). only the first displ bytes of intxt are destroyed; the partial quartet at the end is intact, and can be concatenated with later data and passed to a subsequent call to this routine.
4. if an invalid character is found, retcod=4 and displ=displacement to the bad character. any preceding complete quartets are processed, and outlen=3*(the number of them).

Examples:

In these examples, x and y in outtxt are not the actual output characters, but indicate the form of the output.

```
intxt='abcdefgh',inlen=8: outtxt='xxxyyy',outlen=6,
       retcod=0,displ=8

intxt='abcd efgh',inlen=9: outtxt='xxx',outlen=3,
       retcod=0,displ=4

intxt=' ',inlen=0: outtxt='',outlen=0,retcod=0,displ=0
```

```
intxt='abcdefg=',inlen=8: outtxt='xxxyy',outlen=5,
       retcod=-1,displ=8

intxt='abcdef==',inlen=8: outtxt='xxxy',outlen=4,
       retcod=-2,displ=8

intxt='abcdef=',inlen=7: outtxt='xxxy',outlen=4,
       retcod=-2,displ=7 (this is the exception in case (2).)
    (same result for intxt='abcdef= ',inlen=8)

intxt='abcde',inlen=5: outtxt='xxx',outlen=3,retcod=1,displ=4

intxt='abcdefg',inlen=7: outtxt='xxx',outlen=3,
       retcod=3,displ=4

intxt='abcdef*h',inlen=8: outtxt='xxx',outlen=3,
       retcod=4,displ=6

intxt='abcde===',inlen=8: outtxt='xxx',outlen=3,
       retcod=4,displ=5

intxt='abcdef=*' or 'abcdef=h',inlen=8: outtxt='xxx',outlen=3,
       retcod=4,displ=6 (the "=" is the bad char because it is
       not followed by another "=")
```

*Notes:*

1. intxt argument is modified by this routine. See description of intxt above.

2. Other 3-to-4 encoding schemes may use a different translate table and pad character, and may handle lengths which are not a multiple of 3 differently.

3. This routine assumes the base 64 encoded text contains EBCDIC (not ASCII) printable characters. If the input is ASCII, the caller must convert the input to EBCDIC before calling this routine.

# FILL

This routine sets each byte of an area to a specified character.

**Calling Sequence:** CALL FILL(area,len,char)

**Arguments:**

area is the name of the storage area.

len is the length in bytes of the storage area. If the *len* is 0 or negative, no bytes are changed.

char is the character to use to fill in the area.

**Example:**

```
CALL FILL(RECORD,80,' ')
```

# FILMSG

Get descriptive error text corresponding to a file error condition. Refer to the section "Dynamic Access to Files" for a description of this routine.

# FIXSCR

This subroutine translates the hex characters x'00' to x'3f' to x'40 (blank). Under certain conditions the hex characters between 00 x'00' and x'3f' combine with printable characters to accidentally produce 3270 data stream orders. These will disrupt the screen format when FSIO requests or panel are used. FIXSCR prevents these characters from damaging the screen format.

**Calling Sequence:** CALL FIXSCR(a,len)

**Arguments:**

a is the data to be translated.

len is the number of bytes to be translated (0 or more). length may exceed 256. no action if length is 0 or less.

# FNDALL

This routine finds all the characters of a string that are one of the group of characters specified in *reference string*. All reference characters located will have their relative position in the string returned, as well as the number found.

**Calling Sequence:** CALL FNDALL(string,strlen,refstr,reflen,pos,numpos)

**Arguments:**

string is a character string of length strlen, that is to be verified.

strlen is the length of the string, in the range of 1 to 256.

refstr is the group of character(s) that constitute the set of characters that are to be found in *string.* This character string can be called the *reference string* and the characters *reference characters*.

reflen is the length of the *refstr,* in the range 1 to 256.

pos is an integer array where the relative character positions of matched *refstr* characters are returned. *pos* should be dimensioned to the byte length of *string,* that is *pos(strlen).* If a character is detected in *string,* and is one of the reference characters, its relative position is returned in *pos(i).*

numpos is an integer where the number of relative character positions of matched *refstr* characters are returned.

**Example:**

Given that:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789'
REFLEN --> 10

CALL FNDALL(STRING,STRLEN,REFSTR,REFLEN,POS,NUMPOS)
```

Returns:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789'
REFLEN --> 10
POS    --> 3, 4, 6, 7, 0, 0, 0, 0
NUMPOS --> 4
```

# FNDCHR

This routine finds the first occurrence of any of the characters specified by a *reference character string* in string. If a reference character is located, its relative position in the string is returned.

**Calling Sequence:** CALL FNDCHR (string,strlen,refstr,reflen,pos)

**Arguments:**

| | |
|---|---|
| string | is a character string of length *strlen*, that is to be scanned. |
| strlen | is the length of the string, in the range of 1 to 256. |
| refstr | is the group of character(s) that constitute the range of characters, that is to be searched for, in *string*. This character string can be called the *reference string* and the characters *reference characters*. |
| reflen | is the length of the *refstr*, in the range of 1 to 256. |
| pos | is an integer returned that is set to 0 if none of the characters of string are also in the reference string, *refstr*. If a character is detected in *string*, that is one of the reference characters, its relative position is returned in *pos*. |

**Example:**

Given that:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789b'
REFLEN --> 11

CALL FNDCHR(STRING,STRLEN,REFSTR,REFLEN,POS)
```

Returns:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789b'
REFLEN --> 11
POS    --> 3
```

# FRMTDA

(Fortran G1 only) A call to this subroutine writes blank records throughout the direct access file defined on MUSIC I/O unit number *n.* It can be used to erase all previous contents of the file.

**Calling Sequence:** CALL FRMTDA(n)

# FRSTOR

This subroutine frees main storage for use via an array in an os-mode program (normally FORTRAN). FRSTOR does a FREEMAIN to free a specified amount of storage starting at a specified offset from a given array. See also GTSTOR.

**Calling Sequence:** CALL FRSTOR(amt,a,offset,retcod)

**Arguments:**

| | |
|---|---|
| amt | number of bytes to be freed. This number is always rounded up to a multiple of 8. *amt<=0* is a no-operation. *amt* is rounded up to a multiple of 8 by freemain. |
| a | an array relative to which the storage area is referenced. |
| offset | distance in bytes from the array to the storage area. The allocated area is always on a doubleword boundary. The area must be on a doubleword boundary. |
| retcod | return code:<br>0 successful.<br>1 requested storage not available.<br>2 invalid request or argument value (e.g. area to be freed is not on doubleword boundary).<br>3 not OS mode. |

*Notes:*

1. An attempt to free storage that you did not get, or that has already been freed, can result in job termination by the GETMAIN/FREEMAIN processor in OSTRAP.

2. neither get nor free clears storage. The caller should do this (by calling the ZERO subroutine) if desired.

3. If free storage is fragmented, a request for maximum available storage returns the largest contiguous free area, which may be much less than the total free storage.

# GETID

This routine gets the ownership id from a file name or a data set name.

**Calling Sequence:** CALL GETID(type,name,lname,idout,pos)

**Arguments:**

type — ID type:
- 1 name is a file name, and end of id is indicated by a colon (:), as in userid:name_proper
- 2 name is a UDS data set name, and end of ID is indicated by a period (.), as in USERID.ABC.DEF. also, if the DSNAME does not contain a period, and is length 4 or more, the userid is assumed to be the first 4 characters. Any other value for type sets IDOUT=' ',POS=1.

name — (input) the name to be processed.

lname — (input) max length of the name. The name is ended by a blank or after lname characters, whichever comes first.

idout — (output) 16-char userid from the name, padded with trailing blanks if necessary, or blanks if the name does not start with a 1 to 16 character userid.

pos — (output) integer position of the first character of the name proper, after the userid. Value 1 means no valid userid was present. Value *n* means the name proper starts at the n'th character of the name (counting from 1). n>1 means a valid userid was found.

# GETOP

This subroutine gets the parameters from a character string. It is used by commands /LIST, /SAVE, /PURGE, /INPUT, etc. and by some utility programs.

Parameters in the string are separated by one or more blanks or commas, and may be preceded by leading blanks (but not leading commas - leading commas are considered to be part of the first parameter). Optionally, a command keyword at the beginning of the string may be skipped.

**Calling Sequence:** CALL GETOP(ctl,prmfld,prmlen,optbl,maxop,kwtbl,&err)

**Arguments:**

ctl is an integer containing option bits in the 4th byte:
- x'01' the input string (prmfld) starts with a command keyword which should be skipped. This is done by searching for the first blank and starting the parameter scan after that blank. If no blank is found, there are considered to be no parameters.

(other bits reserved)

prmfld is the input character string containing the parameters. It may have leading and trailing blanks (after the command keyword has been skipped, if x'01' option in *ctl).*

prmlen is the length (0 or more) of the character string.

optbl is the table to hold the parameter information. The first word will be set to the number of parameters

found (0 or more). The rest of the table consists of entries, each 3 words long.

1st word of entry: address of the parameter in the character string.

2nd word of entry: length of the parameter.

3rd word of entry: if the parameter is 1 to 9 decimal digits, this is the numeric value (0 or more). Otherwise the value is negative. The first byte is x'80', or x'80'+n if the parameter matches the keyword with id n in the keyword table *(kwtbl)*. Note: if a keyword in *kwtbl* is all decimal digits, The 3rd info word does not have the numeric value but starts with x'80'+n.

maxop is the maximum number of parameters that *optbl* can hold. the total size of the table is *1+3*maxop* words.

kwtbl is the table of special keywords which may occur as parameters. Each is given a non-zero id number n, which is returned in the *optbl* as described above. Different keywords may have the same id number. Each entry is of variable length:
1st byte: length of keyword, or 0 meaning end of table.
2nd byte: length of minimum abbreviation.
3rd byte: the id number n.
remaining bytes of entry: the keyword.

&err is the alternate return is taken if there are more than *maxop* parameters. The first *maxop* parameters are still put into the table, and the first word is set to *maxop*. For assembler callers, output r15=0 means no error, r15=4 means too many parameters.

# GETRET

This routine returns the return code set by the previous job. This also includes jobs done via the NXTCMD subroutine.

**Calling Sequence:** CALL GETRET(irc)

**Arguments:**

irc is an integer number representing the return code set by the previous job.

# GTSTOR

This subroutine gets main storage for use via an array in an OS-mode program (normally FORTRAN). GTSTOR does a GETMAIN to get a specified amount (or the max amount available) of main storage, and passes back an offset "T", which is the distance in bytes from the start of a given array "A" to the start of the allocated storage. A(T+1) then references the first element of the storage. The array is normally declared as size 1 in the calling program. See also FTSTOR.

**Calling Sequence:** CALL GTSTOR(amt,a,offset,lenget,retcod)

**Arguments:**

amt number of bytes to get. This number is always rounded up to a multiple of 8. *amt=0* requests the maximum storage available.

a — an array relative to which the storage area is referenced.

offset — distance in bytes from the array to the storage area. The allocated area is always on a doubleword boundary. This is set to 0 if there is an error.

lenget — the number of bytes to get(a multiple of 8). If there is an error, *lengot* is set to 0.

retcod — return code:

0 successful.
1 requested storage not available.
2 invalid request or argument value (e.g. amt<0 for get).
3 not OS mode.

*Note:* GTSTOR does not clear storage. The caller should do this (by calling the ZERO subroutine) if desired.

# GULPDF

This subroutine is used to define a file to be processed using GULPRD and/or GULPWR subroutines for efficient reading and writing of large amounts of data.

**Calling Sequence:** CALL GULPDF(fileno {,sblk,filid} ,blksiz,numblk)

**Arguments:**

fileno — specifies the unit number and must be an integer from 1 to 15 inclusive. These numbers correspond to a /FILE statement defining a UDS-type disk data set or file.

blksiz — is the size of each block of data (in bytes) and is limited only by the size of the array to be used for data.

numblk — The program sets *numblk* to the number of blocks of data which can be stored in the data set.

sblk — is optional, it specifies the physical record number (number of the 512 byte block) to be used as the first block of a logical data set. If *sblk* is used then *filid* must be specified also.

Filid — is a logical unit number (specified by the user) by which this logical file is to be referenced, and must be a value from 1 to 32 inclusive: All arguments are integers.

*Notes:*

1. If *sblk* and *filid* are specified, the program can be called more than once for the same *fileno* permitting several logical files to be defined on one data set. If *sblk* and *filid* are not specified, they are assumed to be equal to 1 and fileno, respectively.

2. When the data set is originally allocated, a record size of 128 bytes should be specified. For a file, the record format should be F or U. If the file is to have n blocks of blksiz bytes each, the number of records specified at allocation time should be (with the result of the division truncated before multiplication):

$$4n \frac{(blksiz + 511)}{512}$$

3. The actual amount of disk space used for each logical block is a multiple of 512 bytes. For this reason, very small blksize specifications, or those just slightly larger than multiples of 512 tend to waste disk space.

4. No other I/O statements (direct access or sequential) should be used on data sets used by these routines.

5. The GULP routines can be used only with UDS or files on disk. They cannot be used with tape files.

6. When using GULPWR to write to a new file for the first time, *gaps* must not be left between blocks. For example, you may not write blocks 1,2, and 5 in that order, because blocks 3 and 4 would be a gap. A sequence such as 1,2,3,2,4,5 is allowed. Also, you may not read a block which has never been written. These restrictions do not apply to UDS files.

# GULPRD

This subroutine is used to read a file defined by a call to the GULPDF subroutine.

**Calling Sequence:** CALL GULPRD(filid,blkno,array {,len})

or

CALL GULPRD(fileno,blkno,array {,len})

**Arguments:**

filid specifies the logical unit defined in the call to GULPDF. If it was not specified, *filid* should be the *fileno* specified in the call to GULPDF.

blkno is the number of the logical block at which reading is to begin.

array is the variable into which the data is to be read.

len is the number of bytes of data to be read, and its value must be from 1 to *blksiz*. *blksiz* is the argument supplied with GULPDF.) *Len* is optional and if omitted, *blksiz* is used.

*Notes:*

1. For a given block, if more data is read than had been written, the excess data has an unpredictable value.

2. All arguments are integers, except array which can be any type.

# GULPWR

This subroutine is used to write a file defined by a call to GULPDF. (Arguments follow the same rules as GULPRD.)

**Calling Sequence:** CALL GULPWR(filid,blkno,array {,len})

or

CALL GULPWR(fileno,blkno,array {,len})

**Arguments:**

| | |
|---|---|
| filid | specifies the logical unit defined in the call to GULPDF. If it was not specified, *filid* should be the *fileno* specified in the call to GULPDF. |
| blkno | is the number of the logical block at which writing is to begin. |
| array | is the variable into which the data is to be written. |
| len | is the number of bytes of data to write, and its value must be from 1 to *blksiz*. *blksiz* is the argument supplied with GULPDF.) *Len* is optional and if omitted, *blksiz* is used. |

# HWORD

This subroutine is a FORTRAN INTEGER function which returns the first two bytes of the argument *x*. Your program must declare this function as INTEGER.

**Calling Sequence:** n=HWORD(x)

# ITSFxx

The following are indexed text search (ITS) retrieval subroutines for basic indexing:

| | |
|---|---|
| ITSFID | initialization/dynamic call |
| ITSFIW | initialization/workarea call |
| ITSFOP | open a search set |
| ITSFSS | search string |
| ITSFSW | search word list |
| ITSFOR | order result list |
| ITSFRE | retrieve results |
| ITSFCL | close search |

See the topic "ITS Subroutines" later in this chapter for more information.

# I2C

This routine converts an integer to its EBCDIC character representation.

**Calling Sequence:** CALL I2C(i,string,sublen)

**Arguments:**

| | |
|---|---|
| i | any 4 byte location to be converted to its EBCDIC character representation. |
| string | is the output string where the EBCDIC character representing the original string is returned. The character string representing the number is left justified in *string*. *String* must be at least 12 bytes long. |
| sublen | is a returned value that contains the length of the character string stored in string. |

**Example:**

Given that:

```
I      --> 888

CALL I2C(I,STRING,SUBLEN)
```

Returns:

```
I      --> 888
STRING --> '888bbbbbbbbb'
SUBLEN --> 3
```

# I2X

This routine converts an integer to its hexadecimal character representation.

**Calling Sequence:** CALL I2X (i,hexstr)

**Arguments:**

| | |
|---|---|
| i | any 4 byte location to be converted to its hexadecimal representation. |
| hexstr | is the output string where EBCDIC hexdigits representing the original string are returned. *Hexstr* must be at least 8 bytes long. |

**Example:**

Given that:

```
I      --> 249

CALL I2X(I,HEXSTR)
```

Returns:

```
I      --> 249
HEXSTR --> '000000F9'
```

# KEEPIN

This subroutine is used to keep the contents of the input area for 3270-type workstations only. When a response to a conversational read is entered in the input area, it is not erased when the ENTER key is pressed. (See CLRIN.)

**Calling Sequence:** CALL KEEPIN

# LAND

This subroutine is a FORTRAN INTEGER function which is used to perform a logical *and* of two fullword operands *a* and *b.*

**Calling Sequence:** n=LAND(a,b)

**Example:**

```
N=LAND(ABLE,BAKER)
```

# LBCOMP

This subroutine is a FORTRAN INTEGER function used for comparing bytes. The function returns a value of -1, 0, or 1 corresponding respectively to whether *a* is less than, equal to, or greater than *b.*

**Calling Sequence:** m=LBCOMP(a,ka,b,kb,n)

**Arguments:**

a is one location of bytes.

ka indicates which byte to start with at location *a.* (Bytes are numbered starting at 1, not 0.)

b is another location of bytes.

kb indicates which byte to start with at location *b.* (Bytes are numbered starting at 1, not 0.)

n is the number of bytes to compare. If *n* is less than one, the function returns a value of zero.

**Example:**

```
IF(LBCOMP(ALPHA,1,BETA,3,2)) 10,20,30
```

# LBMOVE

This subroutine is used to copy bytes from one location in main storage to another location.

**Calling Sequence:** CALL LBMOVE(a,ka,b,kb,n)

**Arguments:**

a is the location in main storage of the bytes to be moved.

ka indicates which byte to start with at location *a.* (Bytes are numbered from 1, not 0.)

b is the location in main storage to move the bytes from location *a.*

kb indicates which byte to move to at location *b.* (Bytes are numbered from 1, not 0.)

n is the number of bytes to move. If *n* is less than one, no action is taken.

**Example:**

```
DIMENSION ABLE(20),BAKER(20)
:
CALL LBMOVE(ABLE(5),2,BAKER(2),3,40)
```

# LCOMP

This subroutine is a FORTRAN INTEGER function is used to compare bytes. The function returns a value of -1, 0, or 1 corresponding respectively to whether *a* is less than, equal to, or greater than *b*.

**Calling Sequence:** m=LCOMP(a,b,n)

**Arguments:**

a is one location of bytes.

b is another location of bytes.

n is the number of bytes to compare. If *n* is less than one, the function returns a value of zero.

**Example:**

```
IF(LCOMP(I,J,2)) 10,20,30
```

# LCOMPL

This subroutine is a FORTRAN INTEGER function which is used to transform the single full word argument to its 1's complement.

**Calling Sequence:** n=LCOMPL(a)

# LJUST

This routine left justifies a substring, within a 1 to 256 character string, defined as the first non-blank character from the left and the last non-blank character.

**Calling Sequence:** CALL LJUST(string,strlen,sublen,pad)

**Arguments:**

string is a 1 to 256 character string to be left justified.

strlen is the length of the string, an integer in the range of 1 to 256. When strlen =0 no action is taken by LJUST.

sublen is the length of the character string defined by the first non-blank character from the right.

pad all trailing blanks in the string are converted to the pad character.

**Example:**

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8
PAD    --> 'b'

CALL LJUST(STRING,STRLEN,SUBLEN,PAD)
```

Returns:

```
STRING --> '12345bbb'
STRLEN --> 8
SUBLEN --> 5
```

# LMOVE

This subroutine is used to copy bytes in main storage.

**Calling Sequence:** CALL LMOVE(a,b,n)

**Arguments:**

a is the start of the location in main storage of the bytes to move.

b is the start of the location to receive a copy of the bytes.

n is the number of bytes to move. If *n* is less than one, no action is taken.

# LN

This subroutine gets the length of a character string, not counting trailing blanks.

**Calling Sequence:** n=LN(charvar)

or

n=LN(area,len)

**Arguments:**

charvar is a VS Fortran character variable or character array item. The declared length is found from the extra arglist info passed by VS Fortran.

area is any character string, of length "len".

n is the returned length of the string, not counting trailing blanks. n=LN(area,len) is equivalent to n=LOCATE(area,-len, ' ',-1), i.e. backscan for a nonblank.

*Notes:*

1. The first form, with only 1 argument, is valid only if the argument is a VS Fortran character item. For a non-char item, the length is unknown and n=0 is returned. No error message is issued.

2. If "len" is 0 or negative, n=0 is returned.

# LOCATE

This subroutine searches for the substring within a string. If the string is located *n* is set to the starting position (i.e. byte number, counting from 1) of the substring which matches *str*. If the search is unsuccessful, *n* is set to zero.

LOCATE may be used as a subroutine or as an integer function.

**Calling Sequence:** CALL LOCATE(a,la,str,len,n)

or

n=LOCATE(a,la,str,len)

**Arguments:**

a location of the string to be searched.

la is the length of string *a* It can be specified as a negative or positive integer. If positive, the search for the substring starts at the beginning of *a* and proceeds forward. If *la* is negative, the search starts at the end of *a* and searches backwards. *la* should not be 0.

str is the substring to search for.

len is the length of *str* (in bytes) and must have an absolute value between 1 and 256 inclusive. It can be negative or positive. If *len* is negative, the search is for the first string **not** equal to *str*. Note that if *len* is negative, n=0 means that all substrings of *a* are equal to *str*.

# LOCNAM

This subroutine changes an output file name to make it local to the current directory (if necessary).

**Calling Sequence:** CALL LOCNAM(filnam)

**Arguments:**

filnam is a 64-char file name, that will be used for output or append. This routine adds ".\" (restrict the name to the current directory - or lower) to the front of the name if the name does not already start with any of the following:

| | |
|---|---|
| id: | (a specific userid implies root of that id) |
| \ | (root directory or a specific subdirectory) |
| .\ | (current directory) |
| ..\ | (next higher directory) |
| / | (special names like /input imply the root) |

This is similar to what the /PURGE command does.

For example, suppose an application lets a user specify a file to which some data is to be appended (e.g. mail copy function with the append option). The application should call this routine to adjust the name. If the user specifies file name "abc", this routine would change it to ".\abc". a name like "sam:abc" or "\mydir\abc" would be left as is.

If this routine is not called, there is real danger of a privileged user appending to a public file that he/she does not own, thus corrupting the public file without realizing it.

# LOR

This subroutine is a FORTRAN INTEGER function which is used to perform a logical *or* of the two full word arguments *a* and *b.*

**Calling Sequence:** n=LOR(a,b)

# LSHFTL

This subroutine is a FORTRAN INTEGER function which is used to perform a logical shift left, *k* bits, in the full word argument *a.* If *k* is less than 0, 0 is used. If *k* is greater than 32, 32 is used.

**Calling Sequence:** n=LSHFTL(a,k)

# LSHFTR

This subroutine is a FORTRAN INTEGER function which is used to perform a logical shift right, *k* bits, in the full word argument *a.* If *k* is less than 0, 0 is used. If *k* is greater than 32, 32 is used.

**Calling Sequence:** n=LSHFTR(a,k)

# LXOR

This subroutine is a FORTRAN INTEGER function which is used to perform a logical *exclusive or* of the two full word arguments *a* and *b.*

**Calling Sequence:** n=LXOR(a,b)

# MA2E

This subroutine translates 8-bit ASCII to (MUSIC) EBCDIC. See also: ME2A, A2E, and E2A subroutines.

**Calling Sequence:** CALL MA2E(a,len)

**Arguments:**

a is the data to be translated.

len is the number of bytes to be translated (0 or more). length may exceed 256. No action is taken if length is 0 or less. (This routine is re-entrant.)

*Notes:*

1. This is a one-to-one mapping between the 256 ASCII and 256 EBCDIC characters. When routines MA2E and ME2A are used successively on the same data, there is no net change. It can therefore be used for storing extended ASCII (accented, foreign, and graphics chars, etc.) or binary ASCII on MUSIC in EBCDIC. When the data is later translated back to ASCII, nothing will be lost.

2. Most ASCII control characters are translated correctly. In particular: cr, lf, ht (tab).

3. This translation is very close to that done by ind$file (3270 file transfer). ME2A's translation of EBCDIC 6a to d5 is one exception; ind$file translates it to 7c, making ind$file not one-to-one.

4. Some notable characters:

```
EBCDIC          <--->      ASCII
 4a (cent sign)              e5
 4f (solid vert bar)         7c (split vert bar)
 5f (not sign)               5e (caret)
 6a (split vert bar)         d5
 79                          60 (grave accent)
```

# MCNCAT

This routine can replace VS Fortran's CNCAT# (in IFYCNCAT or AFBCNCAT) concatenation routine, in cases where the Fortran run-time library routines are not available (e.g. MAIL).

Place /INC $SUB:MCNCAT.OBJ in the program or LKED the file. (Note: This routine should *not* be added to the subroutine library.)

**Calling Sequence:** CALL MCNCAT(src1,len1,src2,len2,...,dest,destln)

Entry point CNCAT# is equivalent to MCNCAT. This is equivalent to: dest=src1//src2//...

Strings can be any length, including 0. If *len<=0*, string is considered to be null. The resulting string is truncated or blank-padded to match the target string length. Immediate return if # arguments is not even and >=2.

# ME2A

This subroutine translates (MUSIC) EBCDIC to 8-bit ASCII. See also: MA2E, A2E, and E2A subroutines. (This routine is re-entrant.)

**Calling Sequence:** CALL ME2A(a,len)

**Arguments:**

a is the data to be translated.

len is the number of bytes to be translated (0 or more). length may exceed 256. No action if length is 0 or less.

*Notes:*

1. This is a one-to-one mapping between the 256 ASCII and 256 EBCDIC characters. When routines MA2E and ME2A are used successively on the same data, there is no net change. It can therefore be used for storing extended ASCII (accented, foreign, and graphics chars, etc.) or binary ASCII on music in EBCDIC. When the data is later translated back to ASCII, nothing will be lost.

2. Most ASCII control characters are translated correctly. in particular: cr, lf, ht (tab).

3. this translation is very close to that done by ind$file (3270 file transfer). me2a's translation of EBCDIC 6a to d5 is one exception; ind$file translates it to 7c, making ind$file not one-to-one.

4. Some notable characters:

```
EBCDIC          <--->      ASCII
 4a (cent sign)              e5
 4f (solid vert bar)         7c (split vert bar)
 5f (not sign)               5e (caret)
 6a (split vert bar)         d5
 79 (grave accent)           60 (grave accent)
```

# MODFLD

This routine is used to re-define the attributes of modifiable fields by field number of screens designed using PANEL. The attributes that can be modified are: protection, intensity, hide, and skip. See the PANEL documentation in *Chapter 10. Utilities* for details about these.

**Calling Sequence:** CALL MODFLD(panel,ifld,irc,{iprt,intens,ihide,iskp,fldlen})

**Arguments:**

panel is the panel name whose attributes are to be modified. Note that *panel* must be declared as external in the calling program.

ifld is the field number whose attributes are to be modified.

irc is the return code.
- =0 normal return, instructions followed as specified.
- =1 invalid field number, *ifld* is either less than zero or greater than the highest field number available for this panel.
- =2 skip setting could not be set because the field following the target field did not have an attribute of x'6c' or x'7c' (protected, nondisplay).

iprt is the protection flag:
- <=-1 set to unprotected
- =0 make no change to current field protection status
- >=1 set to protected

intens is the intensity flag:
<=-1 set to low intensity
=0 make no change to current field intensity status
>=1 set to high intensity

ihide is the hide flag:
<=0 make no change to current hide setting.
>=1 set attribute to hide (nondisplay, noprint, nondetectable)
To undo hide, reset the intensity; hide is a form of intensity.

iskp is the skip flag:
<=-1 set to no skip
=0 make no change to current skip status
>=1 set skip
Skip for a field can only be set if the field is immediately followed by a protected, numeric, nondisplay/ nondetectable/noprint field.

fldlen the length of the panel field is returned.

*Note:* *iprt*, *intens*, *ihide*, and *iskp* are optional positional arguments. If you do not wish to choose one of these options, assign a value of 0 (zero). If the option(s) appears at the end of the sequence, it may be left out.

**Examples:**

1. The following shows how to change the intensity of field number 2.

```
    CALL MODFLD(PANEL1,2,IRC,0,1,0,0)
or CALL MODFLD(PANEL1,2,IRC,0,1)
```

2. The following example indicates that field number 3 is to be hidden.

```
    CALL MODFLD(PANEL1,3,IRC,0,0,1)
```

# MODOPT

This routine is used to modify PANEL options previously defined using the PANEL subsystem. The options that can be modified are:

1 upper/lower case translation.
2 PF13-PF24 translated to PF1-PF12.
3 PA2 automatic reshow.
4 PF12 as print screen.
5 help screen availability.
6 the help screen to be displayed (by name).

**Calling Sequence:** CALL MODOPT(panel,irc,icode,iset)

**Arguments:**

panel is the panel name whose attributes are to be modified. Note that PANEL must be declared as external in the calling program.

irc is the return code.
=0 normal return, instructions followed as specified.
=1 invalid icode number, ICODE <=0 or > 5.

icode is option number to be reset.
=1 upper/lower case translation.
=2 PF13-PF24 translated to PF1-PF12.
=3 PA2 automatic reshow.
=4 PF12 as print screen.
=5 help screen available.
=6 help screen to be displayed (by name)

iset is the option setting desired:
<=-1 turn off option.
=0 make no change to current setting.
>=1 turn on the option.
= help panel name declared external in the calling program.

**Examples:**

```
CALL MODOPT(PANEL1,IRC,6,HELP1)

CALL MODOPT(PANEL1,IRC,1,-1)
```

# MOVE

This subroutine copies full words from one location to another. *n* full words starting from location *a* to locations starting at location *b*.

**Calling Sequence:** CALL MOVE(a,b,n {,m})

**Arguments:**

a is the location of the words to be moved.

b indicates the location to move the words.

n is the number of full words. If *n* is less than 1, no action is taken.

m is optional, it specifies the length of each element. If *m* is omitted, 4 is assumed.

# NOCRLF

This subroutine is for ASCII terminals only. A call to this subroutine suppresses the carriage return-line feed which normally occurs at the end of each line of output. (See CRLF.)

**Calling Sequence:** CALL NOCRLF

# NOECHO

This subroutine is only used for 3270-type workstations. A call to this subroutine causes the contents entered to the input area of the screen, when responding to a conversational read, not to be displayed on the output area of the screen when the ENTER key is pressed. (See ECHOIN.)

**Calling Sequence:** CALL NOECHO

# NONCAN

A call to this subroutine makes the program non-cancellable. That is, the /CANCEL command can not be used to terminate the program. Use CALL CANCAN to remove the effect of CALL NONCAN. Note that a call to NONCAN only takes effect after the next conversational read or call to CLSOUT.

**Calling Sequence:** CALL NONCAN

# NOPAUS

A call to this subroutine inhibits the system function of stopping every few lines to give the user a chance to cancel the job that is running. (It is meaningful only with 1050 terminals.) See also PAUSE.

**Calling Sequence:** CALL NOPAUS

# NOSHOW

This subroutine is only used for 3270-type workstations. A call to this subroutine causes the characters being entered to the input area of the screen, when responding to a conversational read, not to be displayed. (See SHOWIN.)

**Calling Sequence:** CALL NOSHOW

# NOTRIN

This subroutine suppresses the translation of terminal input data from the terminal code to EBCDIC. This can be used for special applications on ASCII terminals, where the program requires the original terminal data stream. The translation of terminal output data can be accomplished using the X'41' carriage control. Normal translation can be resumed by calling the subroutine TRIN.

**Calling Sequence:** CALL NOTRIN

# NPRMPT

A call to this subroutine can be used to delete the conversational read prompt message entirely. If however, the workstation is set up to pause periodically during output (for example, the /PAUSE command has been issued from a workstation), a question mark (?) is issued as a prompt. (See PROMPT.)

**Calling Sequence:** CALL NPRMPT

# NSGNOF

A call to this subroutine removes the request to schedule a /OFF next. It can be used to negate the effect of a previous call to SIGNOF.

**Calling Sequence:** CALL NSGNOF

# NXTCMD

A call to this subroutine causes the specified command, or program, to be executed. Depending on the options specified, the command is either run concurrently with the calling program using multi-tasking support or is run after the calling program terminates using program chaining.

**Calling Sequence:** CALL NXTCMD(cmd,len{,opt})

**Arguments:**

cmd is the command string to be executed. If the command string specified is not a valid MUSIC command an attempt is made to schedule the user program with that name. The command or program name part of the string should be in upper case. The second part of the command string is the parameter (PARM field).

len is the total length of the command string and should not exceed 196. If the string is too long, the excess characters at the end are ignored.

opt is an optional bit-string value that controls the execution of the command. Only the bits the low order byte are used. The high order bytes must be zero. The X'80' bit determines if the request is for multi-tasking execution or program chaining.

## Program Chaining

In program chaining the system remembers the command specified in the NXTCMD call and executes it automatically when the calling program terminates. The following options are available.

x'80' Must be zero.

x'40' Not used.

x'20' Not used.

x'10' Not used.

x'08' Not used.

x'04' Not used.

x'02' Not used

x'01' Non-cancelable.

Commands that are processed directly by MUSIC's workstation command scanner are not supported by program chaining. These are /COMPRESS, /CTL, /DISCON, /EXEC, /NS, /PAUSE, /PROMPT, /REQUEST, /RUN, /STATUS, /TIME, /TEXT, /TABIN, /TABOUT, /USERS, /WINDOW.

## Multi-Tasking

When the multi-tasking option is used, the system creates a new session and runs the command immediately. This is referred to as the child task. The original session is called the parent. There are a number of options available that define the relationship of the child and the parent. Since multi-tasking is based on the systems multi-session support, multi-session commands such as /NEXT and /PREVIOUS can be used to switch between child and parent.

x'80' Must be set to 1.

x'40' Parent will wait for child task to end. This option is ignored if the parent or child is a background task (BTRM).

x'20' Delete child task at child end of job.

x'10' Displays job startup messages

x'08' Control of the workstation remains with the parent task once the child is started.

x'04' Hide parent task. Multi-session commands cannot be used to return to parent before child terminates.

x'02' Run task in background (BTRM).

x'01' Make child task non-cancelable.

x'0C' Delete child when parent task terminates.

Examples:

```
CALL NXTCMD('HELP SORT',9,128+64+32)
```

This invokes the HELP facility for topic "sort". The parent task waits, and the new task (help) is deleted when the help program ends.

```
CALL NXTCMD('EDIT FILE1',10)
```

This uses program chaining to call the editor.

# NXTPGM

A call to this subroutine causes the specified program to be executed when the current job terminates. This routine can also be used to schedule an *always* program. An *always* program is one that is run whenever the workstation would otherwise return to command mode and can be used to run command mode replacement programs such as TODO, or provide sophisticated program chaining facilities. A call to this subroutine with a blank name cancels the previous call.

**Calling Sequence:** CALL NXTPGM('name '{,'parm',len,opt})

**Arguments:**

name is the name of the file which contains the job to be executed. The length of *name* can be from 1 to 22 characters. If name is less than 22 characters in length, it must be terminated with a blank. If the option argument is 16 or 17, the file name can be up to 64 characters long and must be terminated with a blank.

parm is a character string which will be passed to the program that is to be executed. The maximum length of parm is 74 characters.

len is the length of the parameter string. If it is not specified, the parameter string must be terminated by a $ sign (which is not part of the actual parm).

opt is one of the following numbers informing the system that the program is:

1 - non-cancelable
4 - an "always" program
5 - both, non-cancelable and always program
16 - long file name (up to 64 characters)
17 - non-cancelable and long file name

*Note:* If you do not wish to pass any *parm* to the program, either specify only the name argument, or specify the 2nd argument as 0. The *parm* field is not used by *always* programs.

**Examples:**

```
CALL NXTPGM('name ')

CALL NXTPGM('name ','parm$')

CALL NXTPGM('name ','parm',len)

CALL NXTPGM('name ','parm',len,1)     (non-cancelable)

CALL NXTPGM('name ',0,0,4)            (always program)
```

# NXWORD

This subroutine gets the next word from a string. **Warning:** This routine makes assumptions about how the TOUC subroutine is coded.

**Usage:**

```
POS=1
DO UNTIL LENOUT=0
  CALL NXWORD(pos,a,lena,out,maxout,lenout,opt)
  ...(PROCESS THE WORD IN "OUT")...
END-UNTIL
```

**Arguments:**

pos indicates the starting position in the string. It should be set to 1 before calling NXWORD, to start at the beginning of the string (i.e. to get the first word). NXWORD sets *pos* to the position of the delimiter (blank or comma) following the end of the returned word, or to *lena+1* if no word found.

a is the string.

lena is the length of the string.

out is an area, of length *maxout*, to receive the word. The word is truncated or blank-filled to match the area. The search starts at position *pos* in the string. Delimiters (blanks and (optionally) commas) are skipped to get to the start of the next word. The word ends at the next delimiter or end-of-string.

maxout is the length of the output area.

lenout is the length of the word. If the output area is too small, the word is truncated to length *maxout*, but *lenout* is set to the true (longer) length. If there are no more words in the string, *lenout* is set to 0 and *out* is set to blanks.

opt is an (optional argument) numerical option value. option bits in the 4th byte are:

```
   x'01'  Treat commas as a delimiter, in addition to blanks.
   x'02'  Convert the word to upper case.
   x'04'  Consider all characters between ( ) as part of the
  keyword.
         e.g.  abc(xyz ttt )
               abc( 1,2)
               abc(xyz(1 2) fred)
```

If omitted, 0 is assumed.

**Examples:**

```
CHARACTER A*10/'ABCD EFG  '/,OUT*8
POS=1
CALL NXWORD(POS,A,10,OUT,8,LENOUT)
  ... SETS OUT='ABCD    ', LENOUT=4, POS=5
CALL NXWORD(POS,A,10,OUT,8,LENOUT)
  ... SETS OUT='EFG     ', LENOUT=3, POS=9
CALL NXWORD(POS,A,10,OUT,8,LENOUT)
  ... SETS OUT='        ', LENOUT=0, POS=11
```

See also: PROCOP, ABBREV, GETOP, and SEP.

# OPNFIL

Dynamically open a file. Refer to the section "Dynamic Access to Files" for a description of this routine.

# PANFLD

This routine queries infomation about PANEL fields.

**Calling Sequence:** CALL PANFLD(PANEL,RC,FLD,ROW,COL,FLDLEN,PROT,
INTENS,HIDE,SKIP,ATTR,TOTLEN)

**Arguments:**

PANEL — is the panel name whose info is to be queried. Note that PANEL must be declared as external in the calling program.

RC — is the return code.
- = 0 normal return, instructions followed as specified.
- = 1 invalid field number, ifield <0 or > than highest field number available for this panel.
- = 3 wrong number of arguements. the first 3 are required but no more than 12 arguments can be present.

The following are only given if FLD=0

- = 4 No field on this row, no parameters updated
- = 5 After the last field of a row, no parameters updated
- = 6 before the first field of a row information returned is about the first field
- = 7 in between fields of a row information returned is about the next field

FLD — is the field number whose info is to be queried. If FLD is 0, then ROW and COL are used to query the field and can be anywhere in the field.

ROW — is the row on the screen of the field.

COL — is starting column of the field on output, or is any column in the field on input if FLD=0.

FLDLEN — is the length of the panel field.

PROT — is the protection flag:
- = 0 Field is unprotected
- = 1 field is protected

INTENS — is the intensity flag:
- = 0 field is low intensity
- = 1 field is high intensity

HIDE — is the hide flag:
- = 0 field is not hidden
- = 1 field is hidden

SKIP — is the skip flag:
- = 0 Field is not skipped
- = 1 field is skiped

ATTR — is the attribute byte of the field.

TOTLEN — is the offset of this field from the beginning of the data portion of the COMMON block.

# PARM

This subroutine is used to retrieve the parameter string which was passed to the calling program by a /PARM statement or by a parameter string specified on the /EXEC command.

**Calling Sequence:** CALL PARM(buf,buflen,prmlen)

**Arguments:**

| | |
|---|---|
| buf | is the receiving area which is blanked out prior to receiving the parameter string. Leading and trailing blanks of the parameter string are not transferred to *buf*. |
| buflen | is the length of the receiving area which can have a maximum length of 256 characters. |
| prmlen | is set (by the routine) to the actual length of the parameter string that is moved to *buf*. If *prmlen* is longer than *buflen* then only the leftmost "buflen bytes of the parameter string are moved to *buf*. |

# PAUSE

A call to this subroutine cancels the effect of a call to NOPAUS.

**Calling Sequence:** CALL PAUSE

# PROCOP

This subroutine processes an option item in the format abc( xyz ).

The following option forms are accepted:

```
abc
abc(xyz)
abc=xyz
abc( xyz )
abc= xyz
```

**Calling Sequence:** CALL PROCOP(a,len,kwlen,subpos,sublen,numval)

**Arguments:**

| | |
|---|---|
| a | is the option string. It should not have leading or trailing blanks. |
| len | is the length of a. |
| kwlen | is the (output) length of keyword part (abc), i.e. length up to first ( or = or end of string. |
| subpos | is the (output) position in a of start of suboption (xyz), or 0 if no suboption. |
| sublen | is the (output) length of suboption (xyz), or 0 if none. |
| numval | is the (output) numeric value of mmm (if mmm is 1 to 8 digit chars), otherwise -1. mmm is xyz (ignoring leading and trailing blanks) if there is a suboption, otherwise mmm is abc. |

**Examples:**

```
CALL PROCOP('123',3,N1,N2,N3,N4) --> 3,0,0,123
CALL PROCOP('12X',3,N1,N2,N3,N4) --> 3,0,0,-1
CALL PROCOP('AB(257)',7,N1,N2,N3,N4) --> 2,4,3,257
CALL PROCOP('AB=257',6,N1,N2,N3,N4) --> 2,4,3,257
CALL PROCOP('AB( 257 )',9,N1,N2,N3,N4) --> 2,5,3,257
CALL PROCOP('AB= 257',7,N1,N2,N3,N4) --> 2,5,3,257
CALL PROCOP('AB( 25X )',9,N1,N2,N3,N4) --> 2,5,3,-1
CALL PROCOP('AB(-257)',8,N1,N2,N3,N4) --> 2,4,4,-1
```

See also: NXWORD, ABBREV.

# PROMPT

Cancels the effect of the NPRMPT subroutine.

**Calling Sequence:** CALL PROMPT

# PURGE

This routine deletes a file from the Save Library.

**Calling Sequence:** CALL PURGE(filename,irc)

or

CALL PURGE(-1,longname,irc)

**Arguments:**

filname — is a file name, with or without the userid prefix. If the name is less than 22 characters long, it must be followed by at least 1 blank. The maximum length is 22 characters.

-1,longname — is a long file name, up to 64 characters. If the name is not the maximum length, it must be followed by a blank.

irc — is an integer return code, 0 meaning the file was deleted successfully. For the meaning of nonzero return codes (file not deleted), see the topic "Dynamic Access to Files" of this guide. Also, HELP is provided with the topic ERRORS. (For example, 30 means file not found, 33 means file in use.)

*Note:* This routine does not preserve R0, R1.

**Example:**

```
CALL PURGE('MYFILE ',K)
```

# QFOPEN

Opens a file and defines a buffer area. Refer to the section "Dynamic Access to Files" for a description of this routine and QFCLOS, QFREAD, QFRBA, QFREW, and QFBKRD.

# REREAD

(FORTRAN (G1) only) - allows rereading records as many times as desired, without the expense of physical I/O. The record can be read with the same or different formats and lists. REREAD must be called only once at the beginning of the FORTRAN program. Thereafter a formatted or list-directed read from unit 99 rereads the record read by the last formatted read. No other I/O statement may appear between the two READ statements.

The FORMAT and list should specify single record input, or the results are undefined. The following is an example of a case that should be avoided:

```
          READ(99,200)I,J
      200 FORMAT(I4)
```

**Calling Sequence:** CALL REREAD

**Example:**

```
*Go
list sample
*In progress
      CALL REREAD
    8 READ(5,1,END=10)I
    1 FORMAT(I1)
      GO TO (2,3),I
    2 READ(99,4)J
    4 FORMAT(1X,I10)
      WRITE(6,5)J
    5 FORMAT('0FORMAT 4 WAS USED, J=',I10)
      GO TO 8
    3 READ(99,6)A
    6 FORMAT(1X,F10.2)
      WRITE(6,7)A
    7 FORMAT('0FORMAT 6 WAS USED, A=',F10.2)
      GO TO 8
   10 STOP
      END
/DATA
1         4
2         4
*End
*Go
sample
*In progress
MAIN   = 000250
003668 BYTES USED
EXECUTION BEGINS

FORMAT 4 WAS USED, J=            4

FORMAT 6 WAS USED, A=         0.04
STOP        0
*End
*Go
```

# RJUST

This routine right justifies a substring, within a 1 to 256 character string, defined as, the first non-blank character from the left and the last non-blank character.

**Calling Sequence:** CALL RJUST(string,strlen,sublen,start,pad)

**Arguments:**

string is a 1 to 256 character string to be right justified.

strlen is the length of the string, an integer in the range of 1 to 256. When strlen =0 no action is taken by RJUST.

sublen is the length of the character string defined by the first non-blank from the left and the first non-blank character from the right.

start this is the starting byte of the substring justified on string.

pad all leading blanks in the string are converted to the pad character, after the text is justified.

**Example:**

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8
PAD    --> 'b'

CALL RJUST(STRING,STRLEN,SUBLEN,START,PAD)
```

Returns:

```
STRING --> 'bbb12345'
STRLEN --> 8
SUBLEN --> 5
START  --> 4
```

# RSTART

A call to this subroutine initializes a general purpose random number generator routine. The basic technique used is a combination of a multiplicative congruential generator and a shift-register generator. Each time a random number is requested, two new numbers are generated and then combined to form the next uniformly distributed number of the sequence (or the number from which a normal or exponentially distributed number is derived).

**Calling Sequence:** CALL RSTART(i,j)

**Arguments:**

i is an integer which becomes the starter of the multiplicative congruential generator sequence. If *i* is zero then the sequence of numbers that it starts is zero, so that CALL RSTART (0,j) provides a pure shift register generator.

j is an integer which becomes the starter of the shift-generator. If *j* is zero then the sequence of numbers that it starts is zero, so that CALL RSTART(i,0) provides a pure multiplicative generator.

After calling RSTART, calls to functions UNI, VNI, RNOR, REXP, IUNI, and IVNI can be made to get the next random number in the sequence.

u=UNI(0) provides a normalized floating point variate uniformly distributed on (0<=u<1).

v=VNI(0) provides a normalized floating point variate uniformly distributed on (-1<=v<1).

r=RNOR(0) provides a normalized floating point variate from the standard normal distribution, with a mean of 0.0 and a variance of 1.0.

a=REXP(0) provides a normalized floating point variate with the standard exponential density e**(-y) where (y>=0).

i=IUNI(0) provides an integer variate uniformly distributed in the range (0<=i<2**31).

i=IVNI(0) provides an integer variate uniformly distributed in the range (-2**31<=i<2**31).

Example:

```
C  DISPLAY 50 RANDOM NUMBERS UNIFORMLY DISTRIBUTED BETWEEN 0 AND 1
      REAL X,UNI
      CALL RSTART(12345,98765)
      DO 10 N=1,50
      X=UNI(0)
10    WRITE(6,20) X
20    FORMAT(1X,F10.5)
      STOP
      END
```

*Notes:*

1. If RSTART(0,0) is used, every number generated will be zero. If CALL RSTART is not used, built-in starting values are used.

2. Because the basic random number is a combination of multiplicative and shift-register generators, the resulting sequence has a very large period (approximately 5 x 10**18). The random variables returned from the RNOR and REXP routines have exactly the required distribution, normal and exponential respectively. Both the RNOR and the REXP procedures are extremely fast, generating a variate in less than twice the time required to generate the uniform variate.

# RVRS

This routine reverses a 1 to 256 character string by swapping the characters end to end.

**Calling Sequence:** CALL RVRS(string,strlen)

**Arguments:**

string is a 1 to 256 character string to be reversed.

strlen is the length of the string, an integer in the range of 1 to 256. When strlen =0 no action is

taken by RVRS.

**Example:**

Given that:

```
   STRING --> 'abcdefgh'
   STRLEN --> 8

CALL RVRS(STRING,STRLEN)
```

Returns:

```
   STRING --> 'hgfedcba'
   STRLEN --> 8
```

# SAVREQ

A call to this subroutine schedules a /SV name command to be scheduled for execution. The command is executed automatically as soon as the program terminates execution, whether it terminates normally or abnormally. The save is not be executed if the job is canceled by the system because of excessive output, or if a call to SIGNOF is made after the call to SAVREQ, or if the job is canceled by a /CANCEL command.

**Calling Sequence:** CALL SAVREQ('name ')

**Arguments:**

name is the file name to be used and must be six characters in length.

# SEP

This subroutine separates out fields separated by 1 or more blanks/commas.

**Calling Sequence:** CALL SEP(a,alen,maxnum,num,pos(i),len(i))

**Arguments:**

| | |
|---|---|
| a | is the string to be processed (input). |
| alen | is the length of the string (input). |
| maxnum | is the size of arrays *pos* and *len* (input). |
| num | is the number of fields found (0 to *maxnum)* (output). |
| pos(i) | is the position of the i'th field (output). (Position 1 is the first character of the string.) |
| len(i) | is the length of the i'th field (output). |

*Notes:*

1. If there are more than *maxnum* fields, the extra ones are ignored.

2. If the string contains only blanks or commas, or if *alen=0, num* is set to 0.

## SETINF

Specify information about a new file to be opened by OPNFIL. Refer to the section "Dynamic Access to Files" later for a description of this routine.

## SHOWIN

A call to this subroutine cancels the effect of a previous call to the NOSHOW subroutine.

**Calling Sequence:** CALL SHOWIN

## SIGNOF

A call to this subroutine schedules a /OFF command which is executed automatically as soon as the program terminates execution, whether it terminates normally or abnormally, unless a call to SAVREQ is made after the call to SIGNOF. (See NSGNOF.)

**Calling Sequence:** CALL SIGNOF

## SRTMUS

Generalized disk sort subroutine. For information see "Sorting Routines" in *Chapter 10. Utilities*.

## SSORT

This subroutine is an efficient means of sorting an array (or any table of contiguous, equal-length entries) into ascending order based on a control field within each entry. Comparison is character-type rather than numeric.

**Calling Sequence:** CALL SSORT(array,num,len,keylen,{keypos})

**Arguments:**

| | |
|---|---|
| array | is the table of entries to be sorted. If *array* is a logical*1 2-dimensional array, the dimensions of it would be (len,num). If *array* is a fullword integer array, the dimensions of it would be (len/4,num). |
| num | is the number of entries in *array*. If *num* is less than 2, the subroutine returns since no sort is required. |

len is the length in bytes of each entry and must be a number from 1 to 256 inclusive.

keylen is the length in bytes if the control field to be used for the sort.

keypos is the starting position (byte number, counting from 1) of the control field within each entry. If *keypos* is omitted, 1 is assumed.

# STATUS

This subroutine returns a formatted line consisting of time of day, current date, service units used, and number of users currently signed.

**Calling Sequence:** CALL STATUS(stline,work)

**Arguments:**

stline a 79 byte character string returned.

work a real*8 array of dimension 4 ( real*8 work(4) ).

*Note:* This routine is re-entrant.

# STOPSK

A call to this subroutine terminates the effect of a /SKIP nnn command entered by the user. That is, lines written to the workstation after the call to STOPSK display regardless of any previous /SKIP nnn command. The normal effect of /SKIP nnn is restored after the first such line begins to display.

**Calling Sequence:** CALL STOPSK

# SYSINE

A call to this subroutine indicates the status of the current job stream input file (default unit 5 from FORTRAN).

**Calling Sequence:** CALL SYSINE(mcod)

**Arguments:**

mcod is set by this subroutine to a value depending on current status:

0 No error conditions were associated with the last read operation from the job stream file.
1 End-of-file was encountered at the line read by last read operation.
2 Probable programmer error: for example, /INCLUDE nesting level exceeds 5, or an improper /INCLUDE command was found on the last read.
3 Loss of file integrity caused by an I/O error or some other problem. This could be caused by trying to read a load module file which was created with record format FC (the default). Load modules should be created with record format F. The MUSIC Systems Support Group should be advised if this error persists.
4 (or higher) File is not accessible. The file does not exist or is defined as private or execute only.

A user program is limited to 10 occurrences of error code 4 before the user's job is automatically terminated.

# SYSINL

This subroutine can be used to find information about the current job stream input file (unit 5 from FORTRAN).

**Calling Sequence:** CALL SYSINL(fn,ln,nest,mcod,k,lln,maxdep)

**Arguments:**

fn is the file name currently being read, returned as 8 characters with the last two always blank.

ln is returned as the line number within the current file of the last line read, with 1 being the first line of a file (line number 0 may occur if the first line has not been successfully read).

nest is returned as the current /INCLUDE nest level, if the user sets nest to zero before calling the subroutine. If the user sets nest to a value between 1 and the current nesting level before calling the subroutine, then information about the specified nesting level is returned. (If nest is outside this range, the results of the call are undefined).

mcod returns the same result as a call to SYSINE.

k returns the flags at the current nest level, as defined for subroutine SYSINR. A value of 256 is added to k if the file is owned by the person running the program.

lln specifies a limiting line number, and if specified as -1, then no limit is set.

maxdep returns the maximum nest level the system can handle.

*Note:* File names beginning with a slash (/) character can be returned in the *fn* argument. The name /INPUT refers to the /INPUT file, /TRANX refers to the alternate input file used with /EXEC and /TRMIN means the conversational read spool file.

# SYSINM

A call to this subroutine causes a message to be printed on I/O unit 6, of the form `**AT LINE nnnn of FILE xxxxxx` where *nnnn* is the current line number of the current input file *xxxxxx*. (This subroutine cannot be called from COBOL or VS Fortran programs as it is not compatible with the OS/MUSIC interface.)

**Calling Sequence:** CALL SYSINM

# SYSINR

This subroutine allows the user to dynamically control the system input file (default unit 5 from FORTRAN).

**Calling Sequence:** CALL SYSINR(fn,ln,lln,nc,k)

**Arguments:**

fn is an eight character file name specified by the user, with the last two characters blank. The special name of '/TRMIN ' is used to request spooled conversational reads. For more information see *Chapter 4. File System and I/O Interface* of this guide.

ln specifies the line number to be read by the next operation.

lln specifies the limiting line number at which an end-of-file condition is to be reached. A value of -1 specifies no limit.

nc is a control argument. If *nc=1,* the nest depth is advanced by one and the named input file is set to be read at line number *ln* (with *ln=1* specifying the first line of the file). When the specified file is completely read (or up to line *lln)* the next read operation reads from the following line in the next higher level; i.e the next line in the file being processed at the time of this subroutine call. If the last read operation resulted in an end-of-file condition (including error conditions) then *nc=1* is assumed to be *nc=0.* You can make successive calls to SYSINR if you wish to set up an input file nest, and then your first read statement reads as specified by the last call. Successive calls, cannot in any case, exceed the maximum nesting depth of 5.

If nc=0 is specified, reading of the current file is terminated and the next read operation reads from line *ln* of file *fn.* If the file name *fn* is specified as numeric 0 then the file name is not changed from its current value. This function could be used to skip lines in the current file, although it is more efficient to skip lines by actually reading them.

k is a variable which specifies control flags allowing specification of the same options allowed on the /INCLUDE command. These options are coded in numbers, and the sum of these option numbers form the contents of variable *k* completely replacing those options currently in effect at the specified level.

| Option | Option Number |
|---|---|
| NEST | 1 |
| EOF | 2 (an end-of-file return is always taken when there is no input left to be read) |
| ERRS | 4 |

*Notes:*

1. A CALL SYSINR(0,0,0,0,0) can be used to prematurely stop the reading from the current file. The next read will then start with the next file in the stack or one specified in a subsequent call to SYSINR.

2. If nc=-99 is specified, all nesting levels are totally cleared.

3. Errors resulting from calls to SYSINR are given only at the time of the next read operation. For example, if an invalid file name is specified, no indication will occur until the program actually attempts to read from that file.

4. Calls to SYSINL and SYSINM after a call to SYSINR without any intervening reads can affect the result obtained from these subroutines.

# SYSMSG

This subroutine may be used to control system messages associated with scheduled commands resulting from calls to subroutines SAVREQ and SIGNOF. If the subroutine is called, all system messages associated with the scheduled command (except the final *End message) are suppressed. Messages issued by the program or processor are not suppressed.

**Calling Sequence:** CALL SYSMSG(1)

# TABS

This subroutine is used to specify input or output tab settings from within a program. Its effect is the same as that of a /TABIN or /TABOUT command. The tab settings remain in effect after the end of the job. The physical TAB stops on the workstation are not set by this routine.

**Calling Sequence:** CALL TABS(num,tab {,&n})

**Arguments:**

num specifies the number of tab positions being set and must be positive if input tabs are to be set, and must be negative if output tabs are to be set.

tab is an integer array containing the column numbers to be used, in ascending order. These are the same numbers that would be used in a /TABIN or /TABOUT command.

n is a statement number to return to in the event of an error. This argument is optional.

# TBAS64

This subroutine converts text to base 64 (3 bytes --> 4 bytes) This is the "base 64" encoding scheme used for e-mail mime, as defined in RFC 1521. It encodes any 8-bit text to printable characters, producing 4 output chars for each 3 input bytes. see the translate table in this routine for the 64 printable characters used. An additional character "=" is used for padding at the end of the output if the input length is not a multiple of 3 (see notes below); this allows the original exact length to be determined when decoding. (This routine is re-entrant.)

See also: FBAS64 - decode from base 64; and UUENC - uuencode-style encode.

**Calling Sequence:** CALL TBAS64(intxt,inlen,outtxt,outlen)

**Arguments:**

intxt input data.

inlen length of input data. If 0 or less, outlen is set to 0 and no other action is taken.

outtxt area to receive output data. Size must be at least (n/3)*4 bytes, where n is inlen rounded up to a multiple of 3.

outlen this routine sets outlen to the length of the output data, including pad characters if any. it is always a multiple of 4.

Output pad characters:

If the input length is not a multiple of 3, the last input triple is padded on the right with binary zeros before it is converted, and the last 1 or 2 bytes of the last output quartet are set to the pad character ("="). If input length is 3m+1, there are 2 pad chars; if 3m+2 there is 1 pad char. this lets the decode routine determine the exact length of the original input.

*Notes:*

1. Other 3-to-4 encoding schemes may use a different translate table and pad character, and may handle lengths which are not a multiple of 3 differently.

2. This routine generates ebcdic (not ascii) printable characters in the output. if ASCII characters are desired, the caller must convert the output to ASCII after calling this routine.

Examples:

```
intxt=x'12', inlen=1:   outtxt="eg==", outlen=4

intxt=x'1234', inlen=2:   outtxt="ejq=", outlen=4

intxt=x'123456', inlen=3:   outtxt="ejrw", outlen=4

intxt=x'12345678', inlen=4:   outtxt="ejrwea==", outlen=8
```

# TBIT

This FORTRAN INTEGER function may be used in conjunction with BITON, BITOFF, and BITFLP. It returns the current value of a bit. Your program must declare this function as INTEGER.

**Calling Sequence:** n=TBIT(a,k)

**Arguments:**

a is the location of the byte.

k is the position of the bit. *k* may have the values 0,1,2,... The first bit position is referred to as bit 0. For example, if *k* is 24, the value of the first bit of the fourth byte of *a* is returned. If *k* is negative, a value of zero is returned.

# TEXTLC

This subroutine turns off the translation of lower case characters to their upper case equivalents for future reads from a workstation during the current job. This command is similar in effect to the /TEXT LC command.

**Calling Sequence:** CALL TEXTLC

# TEXTUC

This subroutine turns on the translation of lower case characters to their upper case equivalents for future reads from a workstation during the current job. This command is similar in effect to the /TEXT UC command. This mode is the default unless changed via a CALL TEXTLC or by a /TEXT LC command.

**Calling Sequence:** CALL TEXTUC

# TIMCON

This subroutine converts the time of day, in the form of a character string or an integer value, from one format to another. The second parameter *arg1* is converted from a format specified in *n1* to a format specified in *n2* and is returned in *arg2*. There are 16 format types listed below. Formats 0 to 13 are character string formats and format 14 and 15 are integer values where time is expressed in seconds and minutes respectively.

When input format type *n1* is 0 (zero), the TIMCON subroutine tries to find a correct time sequence by testing all formats. If a match is found, the correct time is converted into *arg2* in format *n2*.

**Calling Sequence:** CALL TIMCON(n1,arg1,n2,arg2,irc{,itype})

**Arguments:**

n1 is the input format type. The format type can be any number from 0 to 15 as defined below.

| Type (n1/n2) | Format (arg1/2) | Length | Example |
|---|---|---|---|
| 0(arg1 only) | ? | 12 | any below |
| 1 | HH | 4 | 15 |
| 2 | HH?M | 8 | 3 pm |
| 3 | HHMM | 4 | 1512 |
| 4 | HHMM?M | 8 | 312 pm |
| 5 | HH.MM | 8 | 15.12 |
| 6 | HH:MM | 8 | 15:12 |
| 7 | HH.MM?M | 8 | 3.12 pm |
| 8 | HH:MM?M | 8 | 3:12 pm |
| 9 | HHMM.SS | 8 | 1512.55 |
| 10 | HH.MM.SS | 8 | 15.12.55 |
| 11 | HH:MM:SS | 8 | 15:12:55 |
| 12 | HH.MM.SS?M | 12 | 3.12.55 pm |
| 13 | HH:MM:SS?M | 12 | 3:12:55 pm |
| 14 | integer time in seconds | 4 | 54775 |
| 15 | integer time in minutes | 4 | 912 |

arg1 is the input character string if type is 0-13, otherwise for type 14 and 15 an integer*4 is expected. The length of the field must be at least as large as is indicated in the table above. When the input type of 0 is specified, the format *arg1* can be any form including "noon" and "midnight" in English, Dutch, French, German, Italian, Portuguese, and Spanish.

n2 is the output format type (see *n1* above). Type 0 is not allowed.

arg2 is the output character string if type is 1-13, otherwise for 14 and 15 an interger*4 is returned. The length of the field must be at least as large as is indicated in the table above.

irc is the return code describing the conversion. The following return codes are possible:

| | |
|---|---|
| -1 | blank input field |
| 1 | invalid integer in time specification |
| 2 | invalid delimiter |
| 3 | time is not inside 0-2400 hours |
| 4 | invalid am/pm specified |
| 5 | invalid format type *(n1* was not 0-15 or *n2* was not 1-15) |
| 6 | invalid format (wrong number of characters for conversion) or when input type is zero, the input field fit no recognizable format. |

itype (optional) echoes the input type format when types 1-15 are used. When input type is 0, the format type assumed by TIMCON is reported here. *itype=0* indicates that 1 of the keywords, noon or midnight was used.

**Examples**

The following examples using input type 0 and output type 11, 13, and 14.

| sample input using type 0 | output as 13 format | output as 11 format | t in secs 14 format | returned itype |
|---|---|---|---|---|
| 13 | 01:00:00 pm | 13:00:00 | 46800 | 1 |
| 3 pm | 03:00:00 pm | 15:00:00 | 54000 | 2 |
| 512 am | 05:12:00 am | 05:12:00 | 18720 | 4 |
| 12:12 | 12:12:00 pm | 12:12:00 | 43920 | 6 |
| 345.59 | 03:45:59 am | 03:45:59 | 13559 | 9 |
| noon | 12:00:00 pm | 12:00:00 | 43200 | 0 |

# TIMDAT

This routine gets the time of day (by TSTIME) and date (by TSDATE). This routine handles the problem of midnight occurring between the calls to TSDATE and TSTIME. It always returns a valid time and date pair.

**Calling Sequence:** CALL timdat(n,time,m,date1,[date2,]...)

**Arguments:**

n is the type of time wanted (as in TSTIME).

time is the output time of day.

m is the type of date wanted (as in TSDATE).

date1,date2 are the output dates.

**Example:**

```
CHARACTER TIME*8,DATE*16
CALL TIMDAT(5,TIME,1,DATE)
```

# TIMOFF

A call to this subroutine can be used to display a message giving the computer time used (in service units) since TIMON was called. If however, TIMON was not previously called, the time displayed by TIMOFF is the time from the beginning of execution of the program.

**Calling Sequence:** CALL TIMOFF

# TIMON

A call to this subroutine is used to reset the job time counter used in the TIMOFF subroutine.

**Calling Sequence:** CALL TIMON

# TOLC

This subroutine translates a string of characters to lower case.

**Calling Sequence:** CALL TOLC(string,strlen)

**Arguments:**

string is a string of any length to be converted to upper case.

strlen is the length of *string.*

**Example:**

Given that:

```
STRING --> 'ABC123'
STRLEN --> 6

CALL TOLC(STRING,STRLEN)
```

Returns:

```
STRING --> 'abc123'
STRLEN --> 6
```

# TOUC

This subroutine translates a string of characters to upper case.

**Calling Sequence:** CALL TOUC(string,strlen)

**Arguments:**

string is a string of any length to be converted to upper case.

strlen is the length of *string.*

**Example:**

Given that:

```
STRING --> 'abc123'
STRLEN --> 6

CALL TOUC(STRING,STRLEN)
```

Returns:

```
STRING --> 'ABC123'
STRLEN --> 6
```

# TPCLSE

A call to this subroutine cancels the effect of a previous call to the TPOPEN subroutine. This subroutine is for ASCII terminals only.

**Calling Sequence:** CALL TPCLSE

# TPOPEN

This subroutine is for ASCII terminals only. When the print line exceeds the terminal line length, MUSIC usually splits the line into two parts. This feature causes problems when users are directing the output to a paper tape. A call to this subroutine suppresses this line splitting.

**Calling Sequence:** CALL TPOPEN

# TRANSL

This routine modifies characters according to a 256-byte translation table. The machine instruction TR (translate) is used.

**Calling Sequence:** CALL TRANSL(a,len,table)

**Arguments:**

a represents the characters to be modified.

len is the length in bytes. *Len* may have any value. If it is 0 or less, no action is taken.

table is the 256-byte translation table supplied by the caller. For each of the 256 possible values of a byte in the argument *a* the table must have a replacement byte.

# TRIN

This subroutine cancels the effect of a call to the NOTRIN subroutine. Normal terminal input translation is resumed.

**Calling Sequence:** CALL TRIN

# TSDATE

This subroutine returns the current date in one of several forms as specified by the first argument *n.* The date is returned in the remaining argument(s) *x, y...* The calling sequence must contain the correct number of arguments depending on the value of *n:* See also TIMDAT.

**Calling Sequence:** CALL TSDATE(n,x,y...)

**Arguments:**

n is a number from 1 to 14 specifying the format of the date.

1 16-byte printable date: WED MAR 06, 1985
2 8-byte U.S.A. format: mm/dd/yy for example 03/06/85
3 8-byte date: *ddmonyyb* for example 06MAR85b (b represents a blank)
4 *x* = integer year (e.g. 1985),
*y* = integer day of year (e.g. 65)
5 integer day of week, 1 to 7 (Sunday is 1, Monday is 2, etc.)
6 *x* = integer month (e.g. 3),
*y* = integer day of month (e.g. 6),
*z* = integer year (e.g. 1985)
7 8-byte European format: dd/mm/yy for example, 06/03/85
8 8-byte sortable: *yymmddbb* for example 850306bb (b represents a blank)
9 8-byte European, no slashes: *ddmmyybb* for example 060385bb (b represents a blank)
10 4-byte packed decimal: 00yydddC for example hex 0055041C (for Mar.6/85)
11 4-byte sortable binary: yymd for example hex 07C10306 (for Mar.6/1985)
12 10-byte sortable: yyyy/mm/dd for example 1985/03/06
13 8-byte sortable: yyyy/ddd for example 1985/065
14 integer in file system date format: integer day+(integer year-1970)*366

x is the variable name where the date is returned.

y is the second variable name where the date is returned and must be specified if *n* is equal to 4 or 6.

z is the third variable name where the date is returned and must be specified if *n* is equal to 6.

# TSTIME

This subroutine returns time of day or job execution time as specified by *n.* See also TIMDAT.

**Calling Sequence:** CALL TSTIME(n,arg)

**Arguments:**

n is a number from 1 - 8 specifying the format or the time as follows:

1 8-byte time of day in the form HH.MM.SS for example, 15.04.09
2 integer time of day in seconds
3 integer job execution time since the beginning of execution, in units of 1/100 service unit.
4 Integer time of day in units of 1/300 seconds.
5 8-byte time of day in the form HH:MM:SS for example 15:04:09
6 11-byte time of day in the form HH:MM:SS.TH, where TH is hundredths of seconds, for example 15:04:09.73
7 8-byte time-of-day clock, as a 64-bit unsigned integer, obtained by executing a Store Clock (STCK) instruction. Note that the time-of-day clock is the elapsed time since a fixed date in the past.
8 Time-of-day clock, converted to number of microseconds by shifting right 12 bits, as an 8-byte REAL*8 normalized floating-point value. Obtained by a Store Clock (STCK) instruction.

arg is the variable name where the time is returned.

# TSUSER

This subroutine returns information about the user as specified by the first argument *n.*

**Calling Sequence:** CALL TSUSER(n,arg)

or

CALL TSUSER(n,arga,argb) (for n=11 only)

**Arguments:**

n is a number from 1 to 15 indicating the type of information required.

1 integer workstation type:
0 batch
1 2741,3767 EBCD code
2 2741,3767,CMCST Correspondence code
3 TTY, narrow carriage (72 characters or less)
4 TTY, wide carriage (more than 72 characters)
5 1050
6 3270
99 other
2 First 8 characters of the user's userid. If the userid is longer than 8 characters, the 8th character is returned as plus sign (+) to indicate truncation. See also n=8, 10.
3 4 character terminal number (always "01 ".) *nn*
4 integer TCB number (session number)
5 terminal primary line length (132 for batch)
6 terminal line speed as preset by installation, in bits/sec (0 for batch)
7 1 if terminal accepts 3270 data streams, 0 otherwise. For example, for PCWS, n=7 returns 1 even though n=1 may return 4.
8 16-character userid of the user. It ends in trailing blanks if the userid is less than 16 characters long.
9 Length of the file ownership part of the userid (1 to 16). This is the length of the userid without the subcode, if any.

10 16-character file ownership id of the user. This is the userid without the subcode, if any.
11 3270 screen size: number of rows is returned in arga (e.g. 24); number of columns is returned in argb (e.g. 80).
12 4 bytes of workstation (terminal) information is returned in arg. The first 2 bytes are as set by call to Q3270 subroutine at sign-on, giving 3270 characteristics:
1st byte: all 0 if batch or not 3270-capable.
bit x'40': PCWS connected via an ASCII line.
bit x'20': NET3270.
bit x'10': 3270 using a protocol convertor (e.g. 7171).
bit x'08': 16 colors.
bit x'04': standard 7 colors (extended color).
bit x'02': monochrome, or only the base 4 colors.
2nd byte:
bit x'80': blink (extended attribute) is supported.
bit x'40': reverse video (extended attribute) supported.
bit x'20': underline (extended attribute) is supported.
bit x'08': reply mode: extended field mode supported.
bit x'04': reply mode: character mode supported.
bit x'02': workstation allows entry of DBCS SO/SI chars.
3rd and 4th bytes: reserved.
13 Integer national language number in effect.
=1 English
=2 French
=3 Kanji (Japanese)
=4 Portuguese
=5 Spanish
=6 German
14 Integer national language display mode. Test result by looking at bits.
= 1 if in double byte display mode (ex: Kanji)
15 Integer userid type number.

arg is the variable name where the information is returned.

# UUDEC

This subroutine converts text from UUENCODE form (4 bytes --> 3 bytes). This is the UUENCODE encoding scheme used by the UNIX and DOS utility uuencode/uudecode. It encodes any 8-bit text to printable characters, producing 4 output chars for each 3 input bytes. see the translate table in this routine for the 64 printable characters used.

This routine decodes the printable data, back to the original data. (This routine is re-entrant.)

See also: UUENC - encode; TBAS64 - encode to base 64 (mime); and FBAS64 - decode from base 64 (mime).

**Calling Sequence:** CALL UUDEC(intxt,inlen,outtxt,outlen,retcod,displ)

**Arguments:**

intxt input data (encoded as by uuencode utility). It is assumed to start on a quartet (group of 4 bytes in the encoded text) boundary. Note: intxt is destroyed by this routine (tranlated in place), so the caller should pass a copy of the original data, if it is to be used again. intxt is destroyed only up to (not including) the first blank or invalid character or to the end of the last full input quartet.

inlen — length of input data. If 0 or less, this routine sets outlen=0, retcod=0, and displ=0 and no other action is taken.

outtxt — area to receive output (decoded) data. It must be large enough to hold the output data. Maximum possible output is ((inlen+1)/4)*3 bytes, where / is integer divide i.e. discard the remainder.

outlen — this routine sets outlen to the length of the output data. This is a multiple of 3. outlen is always the number of bytes stored into outtxt.

retcod,displ — the routine sets these integer arguments to indicate various cases, errors, and ending conditions. retcod is a return code and displ is a displacement within intxt, usually indicating how many input characters have been processed. Scanning stops when a blank is found, or an invalid character is found; this we call the "scan end".

1. if the scan end immediately follows a quartet, retcod=0 and displ=displacement to scan end i.e. 4*(number of quartets processed). outlen=3*(number of quartets processed).
2. if scan end is within a quartet, retcod=number of of chars in last quartet before the scan end i.e. 1 to 3, and displ=displacement to end of the last complete quartet. outlen=3*(number of complete quartets processed). Only the first displ bytes of intxt are destroyed; the partial quartet at the end is intact, and can be concatenated with later data and passed to a subsequent call to this routine.
3. if an invalid character is found, retcod=4 and displ=displacement to the bad character. any preceding complete quartets are processed, and outlen=3*(the number of them).

*Notes:*

1. intxt argument is modified by this routine. See description of intxt above.

2. this routine assumes the encoded text contains EBCDIC (not ASCII) printable characters. If the input is ASCII, the caller must convert the input to EBCDIC before calling this routine.

# UUENC

This subroutine converts text to uuencode form (3 bytes --> 4 bytes). This is the encoding scheme used by the UNIX and DOS utility UUENCODE. It encodes any 8-bit text to (ebcdic) printable characters, producing 4 output chars for each 3 input bytes. see the translate table in this routine for the 64 printable characters used. (This routine is re-entrant.)

See also: UUDEC - decode; TBAS64 - encode to base 64 (mime); FBAS64 - decode from base 64 (mime).

**Calling Sequence:** CALL UUENC(intxt,inlen,outtxt,outlen)

**Arguments:**

intxt — input data.

inlen — length of input data. If 0 or less, outlen is set to 0 and no other action is taken. Note: if inlen is not a multiple of 3, the input data is considered to be extended at the end with 1 or 2 binary 0 bytes (x'00'), so as to be a multiple of 3, and the original input length is not recorded in the encoded data.

outtxt — area to receive output data. Size must be at least (n/3)*4 bytes, where n is inlen rounded up to a multiple of 3.

outlen this routine sets outlen to the length of the output data. It is always a multiple of 4.

# VERALL

This routine performs a variation of the VERIFY routine. It verifies that all the characters of a string are one of the group of characters specified by a *reference character string*. All non-reference characters located will have their relative position in the string returned, as well as a count of all mismatches.

**Calling Sequence:** CALL VERALL(string,strlen,refstr,reflen,pos,numpos)

**Arguments:**

string is a character string of length *strlen* that is to be verified.

strlen is the length of *string.*

refstr is the group of character(s) that constitute the range of acceptable characters that can be found in *string.* This character string can be called the *reference string* and the characters *reference characters*.

reflen is the length of the *refstr* in the range of 1 to 256.

pos is an integer array where the relative character positions of all *non-refstr* characters are returned. *pos* should be dimensioned to the byte length of *string,* that is *pos(strlen).* If a character is detected in *string*, that is not one of the reference characters, its relative position is returned in *pos* (i).

numpos is an integer where the number of relative character positions of all non -*refstr* characters is returned.

**Example:**

Given that:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789'
REFLEN --> 10

CALL VERALL(STRING,STRLEN,REFSTR,REFLEN,POS,NUMPOS)
```

Returns:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789'
REFLEN --> 10
POS    --> 1, 2, 5, 8, 0, 0, 0, 0
NUMPOS --> 4
```

# VERIFY

This routine verifies that all the characters of a string are one of the group of characters specified by a *reference character string*. If a non-reference character is located, its relative position in the string is returned.

**Calling Sequence:** CALL VERIFY(string,strlen,refstr,reflen,pos)

**Arguments:**

string — is a character string of length *strlen* that is to be verified.

strlen — is the length of the string, in the range of 1 to 256.

refstr — is the group of character(s) that constitute the range of acceptable characters that can be found in *string.* This character string can be called the *reference string* and the characters *reference characters*.

reflen — is the length of the *refstr,* in the range of 1 to 256.

pos — is an integer returned that is set to 0 if all characters of string are also in the reference string *refstr.* If a character is detected in *string,* that is not one of the reference characters, its relative position is returned in *pos.*

**Example:**

Given that:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789b'
REFLEN --> 11

CALL VERIFY(STRING,STRLEN,REFSTR,REFLEN,POS)
```

Returns:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789b'
REFLEN --> 11
POS    --> 5
```

# WORD

This routine parses a string by isolating substrings (words) delimited by a specified character (usually a blank). It returns the number of words or substrings found, as well as their positions and lengths.

**Calling Sequence:** CALL WORD(string,strlen,number,pos,lens,delim)

**Arguments:**

string — is a character string to be parsed of length *strlen.*

strlen is the length of the string. When *strlen* =0 no action is taken by WORD.

number is the number of words found in the string.

pos is an array where the relative positions of the words found in *string* are returned. The array dimension should be as large as the expected number of words to be found in *string*. For example, a string of length 80 can only have 40 individual words since the delimiters would occupy 40 bytes.

lens is an array where the relative lengths of the words found in *string* are returned. The array dimension should be as large as the expected number of words to be found in *string*. For example a string of length 80 can only have 40 individual words since the delimiters would occupy 40 bytes.

delim is a one byte string (usually a blank specified as ' ') which serves as the word delimiters. WORD scans *string* and reports, as words, the substrings separated by the delim characters. Consecutive multiple occurrences of the delim character are ignored.

**Example:**

Given that:

```
STRING --> 'Mary had a little lamb.  '
STRLEN --> 25
DELIM  --> ' '

Note: POS and LEN are dimensioned to 12, for this example.

CALL WORD(STRING,STRLEN,NUMBER,POS,LEN,DELIM)
```

Returns:

```
STRING --> 'Mary had a little lamb.  '
STRLEN --> 25
NUMBER --> 5
DELIM  --> ' '
POS    --> 1 6 10 12 19
LEN    --> 4 3  1  6  5
```

# XGCOFF

(Fortran G1 only) A call to this subroutine cancels the effect of a previous call to the XGCON subroutine.

**Calling Sequence:** CALL XGCOFF

# XGCON

(Fortran G1 only) A call to this subroutine permits use of an extended form of G format conversion which permits free format input. The extended form remains in effect until a subsequent call to XGCOFF is made.

In the extended form, the field width specified is used as a maximum field width. However, the field width is considered smaller if the number ends before the specified field width. If no number is found within the

specified field width, then it is set to 0.

**Calling Sequence:** CALL XGCON

**Example:**

This facility allows an input line to be scanned totally in free format or parts can be in fixed format as well, as in the following example.

```
    CALL XGCON
  1 READ(9,2)A,I
  2 FORMAT(2G80.0)
```

The format statement is interpreted as meaning that the two numbers expected may appear anywhere on the data statement, as long as they are separated by at least one blank. Any number not found on the data statement is assumed to be zero.

# X2C

This routine converts hexadecimal strings to character equivalents (EBCDIC).

**Calling Sequence:** CALL X2C(hexstr,hexlen,string,irc,badchr)

**Arguments:**

hexstr — is a character string of length *hexlen* which has in it only hexadecimal digits 0-9 and A-F. These must be paired, e.g. F1F2F3.

hexlen — is the length of the *hexstr*. This length must be an even integer.

string — is the character string to be returned, in the range of 1 to 256. The string should be at least half the length of *hexstr*.

irc — is the return code describing the conversion.
- irc=-1 the HEXSTR length was 0.
- irc=0 conversion was successful.
- irc=1 an illegal character was found in the string.
- irc=3 conversion not possible, odd number of hex digits were found.

*Note:* When *irc* is not 0 *string* remains unchanged by X2C.

badchr — a one byte variable where the bad character (characters not in the range A-F, 0-9) is returned when *irc=1*.

**Example:**

**Given that:**

```
HEXSTR --> 'C1C6F3F4F5F6F7F8F960'
HEXLEN --> 10

CALL X2C(HEXSTR,HEXLEN,STRING,IRC,BADCHR)
```

**Returns:**

```
HEXSTR --> 'C1C6F3F4F5F6F7F8F960'
HEXLEN --> 10
STRING --> 'AF123456789/'
IRC    --> 0
BADCHR --> 'b'
```

# X2I

This routine converts a hexadecimal character string to a four byte integer value.

**Calling Sequence:** CALL X2I(hexstr,hexlen,i,irc,badchr)

**Arguments:**

hexstr — is a character string of length *hexlen* which has in it only hexadecimal digits 0-9 and A-F.

hexlen — is the length of the *hexstr*. This length must be an integer less than or equal to 8 bytes.

i — is the integer to be returned.

irc — is the return code describing the conversion.
irc=-1 the HEXSTR length was 0.
irc=0 conversion was successful.
irc=1 an illegal character was found in the string.
irc=2 conversion not possible, length of *hexstr* was greater than 8.

*Note:* If *irc* is not 0, then: i is set to zero.

badchr — a one byte variable where the bad character (characters not in the range A-F, 0-9) is returned when *irc=1*.

**Example:**

Given that:

```
HEXSTR --> 'E0E'
HEXLEN --> 3
```

```
CALL X2I(HEXSTR,HEXLEN,I,IRC,BADCHR)
```

Returns:

```
HEXSTR --> 'E0E'
HEXLEN --> 3
I      --> 3598
IRC    --> 0
BADCHR --> 'b'
```

# ZERO

This subroutine sets bytes of main storage to zero. The user should use this routine to zero large sections of storage (eg. arrays with more than 500 elements).

**Calling Sequence:** CALL ZERO(a,m {,n})

**Arguments:**

a is the starting location of the bytes in main storage.

m is the number of words to turn to zero. If *m* is less than 1 no action is taken.

n is the length of each word. If *n* is omitted, 4 is assumed so that the subroutine sets *m* full words of main storage to zero.

# Dynamic Access to Files

FORTRAN programs on the MUSIC system can read and write files without having to predefine the file names on /FILE statements. This dynamic style of access allows more program flexibility. For example, at execution time a program can prompt the user to specify the name of a file to be read or written. Input/output is done using the standard FORTRAN statements (READ, WRITE, REWIND, ENDFILE, etc.)

Dynamic access is controlled by four system subroutines: OPNFIL (open a file), CLSFIL (close a file), SETINF (specify information about a new file to be created), and FILMSG (get error message text corresponding to an error number). Some usage examples are given at the end of this article.

## OPNFIL Subroutine (Open a File)

This routine associates a file name with a FORTRAN unit number (1 to 15 for FORTRAN G1, or 1 to 99 for VS FORTRAN. Both the file name and the unit number are specified by the user. Various option keywords may also be specified, indicating a new or existing file, read or write access, etc. The system attempts to open the file, and passes a return code back to the user. A zero return code indicates a successful open, and the user may then do FORTRAN I/O operations on the unit number, just as if the unit had been defined by a /FILE statement.

An 8-character MVS DDname (data definition name) can be specified instead of a unit number. In that case, the DDname is added to the job's DDname table (if not already there) and it is associated with the SL file. This allows programs running in OS simulation mode (such as PL/I, COBOL and VS ASSEMBLER programs) to access files dynamically. OPNFIL gives error code 31 if it is unable to define the DDname because the DDname table is full.

When a program running in OS simulation mode calls OPNFIL with a unit number, the corresponding FORTRAN DDname (FTnnF001) is also defined, in addition to the unit number. This feature allows VS FORTRAN programs to use OPNFIL. Refer to the section on VS FORTRAN for further information.

## CLSFIL Subroutine (Close a File)

The CLSFIL routine closes a unit number previously opened by OPNFIL. The system closes the corresponding file, and the unit number becomes undefined. The unit may be redefined by a later call to OPNFIL. CLSFIL gives a return code to the user. Zero indicates a successful close.

A 8-character MVS DDname can be specified instead of a unit number. This would be a DDname previously defined by a call to OPNFIL.

## SETINF Subroutine (Set Information for a New File)

When a new file is created by OPNFIL, the attributes of the file must be supplied. These include file size, record length, record format, and access control (public, private, etc.) They are specified by calling SETINF before calling OPNFIL. SETINF stores the information in a common block which is then available to OPNFIL.

# FILMSG Subroutine (Get Error Description)

When an open or close is unsuccessful, a nonzero return code indicates the reason for the error. The error code is typically a 2-digit number. If the program wishes to display an error message at this point, descriptive text corresponding to the error code can be obtained by calling FILMSG. Optionally, FILMSG can display a message itself (including the unit number) and terminate the job.

## Calling Sequences

Some of the arguments may be omitted, as indicated in the descriptions below. Note that all numerical arguments are of type INTEGER*4.

**Calling Sequence:** CALL OPNFIL(unit,retcod,'shortname ','options.')

or

CALL OPNFIL(unit,retcod,-1,'longname ','options.')

or

CALL OPNFIL(UNIT,retcod,TYPE)

**Arguments:**

unit — FORTRAN unit number (1 to 99), or an 8-character DDname enclosed in single quotes. For FORTRAN G1, the unit number must be 1 to 15.

retcod — Error number returned by the routine. Zero means successful open.

'shortname ' — The name of the file to be opened. It may include the userid prefix (usrd:). A blank must follow the name, unless it is the maximum length (22 characters). Special names '&&TEMP ' (temporary new file) and '/INPUT ' (MUSIC Input File) may be used. If a temporary file is to be kept, use the RENAME or REPL option when closing it.

'longname' — Same as shortname except that the maximum length of the file name is 64 characters (can include directory prefixes). The option "-1" must be present in the calling sequence before the longname.

'options.' — (This argument is optional.) This is a character string containing option keywords. The keywords are separated by one or more blanks and the last one is followed by a period. They are described below. If this argument is omitted, 'OKOLD RDOK.' is assumed, i.e. an existing file is to be read.

TYPE — A special unit identifier may be specified instead of a file name. The allowable values are:

TYPE = 0 Undefined file.
5 Input stream (card reader if batch).
6 Printed or displayed output.
7 Punched output.
8 Dummy file (read gives EOF, writes ignored).
9 Conversational reads.
10 Special holding file (output).

**Calling Sequence:** CALL CLSFIL(unit,retcod,'options.')

or

CALL CLSFIL(unit,retcod,'options.','newname ')

**Arguments:**

unit — FORTRAN unit number (1 to 99), or an 8-character DDname enclosed in single quotes. For FORTRAN G1, the unit number must be 1 to 15.

retcod — Error number returned by the routine. Zero means successful close.

'options.' — (This argument is optional.) Keyword options, in the same format as for OPNFIL. The keywords for close are described below.

'newname ' — This optional last argument is used only when the RENAME or REPL option is specified. It is the new name to be given to the file.

**Calling Sequence:** CALL SETINF(PRIMSP,SECSP,LRECL,RECFM,'options.')

**Arguments:**

PRIMSP — Initial space to be allocated for the new file, in units of 1K = 1024 bytes. If the argument is omitted or 0, 32 is assumed.

SECSP — Secondary space to be allocated whenever more space is needed in the file. It is specified either as an absolute amount (in units of 1K = 1024 bytes) or as -n, meaning n% of the existing space. If the argument is omitted or 0, 50% is assumed.

LRECL — Logical record length of the file, 0 to 32760. If omitted, 80 is assumed. Specify 0 if the record format is V or VC.

RECFM — Record format, as a 2-character item:
'FC' Fixed-length compressed (the default).
'F ' Fixed-length uncompressed.
'VC' Variable-length compressed.
'V ' Variable-length uncompressed.
'U ' Undefined.
If the argument is omitted or 0, 'FC' is assumed.

'options.' — Keywords defining the access control options for the file, in the same format as the options for OPNFIL. The keywords are described below. If omitted, 'PRIV.' (private file) is assumed, i.e. only the owner may read or write the file.

**Calling Sequence:** CALL FILMSG(errcod,buffer,length)

or

CALL FILMSG(errcod,unit)

**Arguments:**

errcod — An error number returned by OPNFIL or CLSFIL.

buffer — An array to receive the error description.

length — The length of BUFFER, in bytes. A length of 70 or more is adequate, but any length can be

specified. Unused bytes are set to blanks by the routine.

unit — If FILMSG is called with only two arguments, the second is assumed to be the unit number associated with the error. If *errcod* is nonzero, a message is displayed (containing the error number, unit number, and error description) and the job is terminated. If *errcod* is 0, no message is displayed and the job continues.

## Option Keywords for Open

OKOLD — The file may already exist.

OKNEW — A new file may be created. SETINF must be called to define the attributes of the new file. Note: at least one of OKOLD, OKNEW must be used.

RDOK — Read operations will be allowed.

WROK — Write operations will be allowed.

APPOK — Append (adding to the end of the file) will be allowed. The file will be positioned to the end of existing data. WROK must also be used.

POSEND — Position to the end of the file's data.

ENQSHR — Force enqueue for shared control of the file. Normally shared control is used only when WROK is not specified.

ENQEXCL — Force enqueue for exclusive control of the file.

## Option Keywords for Close

RLSE — Release any unused space. This option is recommended if writes were done to the file, especially when creating a new file.

RENAME — Rename the file to the name specified in the 'newname' argument.

REPL — Rename the file to the name specified in the 'newname' argument. This is similar to RENAME, except that if a file already exists with the new name, it is deleted.

DEL — Delete the entire file.

TCLOSE — Specifies that the CLOSE function is to be performed but that the file is to remain open for further requests. The current end of data pointer is updated in the file's directory. RLSE is the only other option that can be given when TCLOSE is used.

CTAG — Set a new tag field when the file is closed.

*Note:* For RENAME and REPL, new access control options are used for the new file, and these should be specified by calling SETINF before calling CLSFIL. In this case, the first 4 arguments of SETINF may be specified as 0.

## Keywords for SETINF Access Control Options

Access is defined for two classes of users: (1) an owner of the file (the user's userid matches the file's userid), and (2) non-owners (all other users). The default is a *private* file, meaning that an owner has unrestricted access while non-owners have no access at all. The suffix "(OWN)" indicates that the keyword applies to access by an owner.

| | |
|---|---|
| PUBL | A publicly readable file. Non-owners may read the file but not modify it. Also, the file is placed in the common index (see COM below). |
| PRIV | A private file (the default). Non-owners can neither read nor write the file. |
| SHR | Non-owners may read the file. |
| COM | The file is placed in the common index, so that non-owners can refer to it without having to include the file's userid in the file name. |
| RD | Same as SHR. |
| NORD | Non-owners may not read the file. |
| WR | Non-owners may modify the file. |
| NOWR | Non-owners may not modify the file. |
| XO | The file is execute-only for non-owners. Execute-only means that the file may be executed as a program, but not read as data. |
| AO | The only type of write access for non-owners is append. This means that non-owners may add data to the end of the file, but not overwrite existing data. This is useful for *log* files. |
| RD(OWN) | Owners may read the file. |
| NORD(OWN) | Owners may not read the file. |
| WR(OWN) | Owners may modify the file. |
| NOWR(OWN) | Owners may not modify the file. |
| XO(OWN) | The file is execute-only for owners. |
| AO(OWN) | The only type of write access for owners is append. |

The following are assumed by default: NORD, NOWR, RD(OWN), WR(OWN). The specification 'PUBL.' is equivalent to 'RD COM.'

## Common Blocks Used

Several named common blocks are used for communication among the calling program and the system subroutines. For most applications, the user need not be aware of them.

| | |
|---|---|
| /FILINF/ | (20 bytes) This area contains information about a new file to be created (set by SETINF |

subroutine), and also receives information when an existing file is opened.

/FILTAG/ (64 bytes) This is the 64-character tag field. To assign a tag to a new file, place the tag in this area before calling OPNFIL, but after calling SETINF. Any call to SETINF zeros this area. Also, when an existing file is opened, its tag is put here.

/FILEFP/ (10 bytes) End-of-file information is placed here when a file is opened.

/FILRGX/ (12 bytes) When OPNFIL or CLSFIL is called, the first 12 bytes of the system request argument (after the request is issued) are placed here. The 10th byte has bit X'80' on whenever a new file is created. After a close with RENAME or REPL, this bit is off if an existing file was replaced.

## Error Codes and Descriptions

```
 1    END OF DATA SET ENCOUNTERED
 2    INCORRECT LENGTH
10    INVALID REQ
11    INVALID REQ PARAMETER
12    FILE NAME INVALID
19    INVALID ARGUMENTS IN CALL TO SERVICE SUBROUTINE
20    TOO MANY OPEN FILES
21    NOT YOUR LIBRARY
22    NOT YOUR FILE
23    VIOLATION OF WRITE RULE
24    ATTEMPT TO READ BEYOND END OF WRITTEN INFO
25    WRITE THEN READ SEQ INVALID
26    YOUR USERID CANNOT CREATE FILES ACCESSIBLE BY OTHERS
27    YOUR USERID CANNOT CREATE FILES IN THE COMMON INDEX
30    FILE NOT FOUND
31    DDNAME NOT FOUND
      (For a call to OPNFIL to define a DDname, error 31
      means there is no more room in the DDname table.)
32    FILE ALREADY EXISTS
33    FILE IN USE
34    COMMON NAME USED BY SOMEONE ELSE
35    UNIT NUMBER NOT DEFINED
36    SUBDIRECTORY DOES NOT EXIST
40    SPACE QUOTA EXCEEDED FOR THIS USERID
41    SPACE QUOTA EXCEEDED FOR THIS FILE
42    CANNOT ADD SPACE TO THIS FILE
43    REQUESTED ACCESS OR OPERATION NOT ALLOWED
44    REQ BEYOND EXTENT OF FILE
45    FILE RECFM NOT DEFINED
46    FILE CANNOT BE READ SEQUENTIALLY
47    INSUFFICIENT SPACE FOR BUFFER ALLOC
48    MIN RECORD LEN IS 80 FOR THIS FILE TYPE
50    FILE NOT ONLINE
51    NOT ENOUGH FREE DISK SPACE
52    NOT ENOUGH FREE DISK SPACE (INDEX)
60    RD I/O ERROR IN FILE
61    WR I/O ERROR IN FILE
62    RD I/O ERROR IN SYSTEM AREA
63    WR I/O ERROR IN SYSTEM AREA
```

```
64    INDEX IN ERROR
65    HEADER IN ERROR
66    MAP INTEGRITY ERROR
67    INDEX/HEADER MISMATCH
70    SYSTEM FILE ERROR
```

## Examples

1. This example opens an existing file JULY.DATA as unit 1, reads it, and closes it.

```
CALL OPNFIL(1,K,'JULY.DATA ')
CALL FILMSG(K,1)
READ(1,...   (read the file)
...
CALL CLSFIL(1,K)
```

2. This sample program opens a new temporary file as unit 12, with a size of 100K, and writes some data to it. It then closes the file, renaming it to a file name supplied by the user. Any unused space is released. The new file is made public.

```
      LOGICAL*1 NAME(22)
      INTEGER UNIT/12/
      CALL SETINF(100)
      CALL OPNFIL(UNIT,K,'&&TEMP ','OKNEW WROK.')
      CALL FILMSG(K,UNIT)
      WRITE(UNIT,10)
10    FORMAT('DATA1...'/'DATA2...')
      WRITE(6,20)
20    FORMAT(' ENTER NAME OF NEW FILE')
      READ(9,30) NAME
30    FORMAT(22A1)
      CALL SETINF(0,0,0,0,'PUBL.')
      CALL CLSFIL(UNIT,K,'RLSE RENAME.',NAME)
      CALL FILMSG(K,UNIT)
      WRITE(6,40)
40    FORMAT(' SAVED')
      STOP
      END
```

# QFOPEN, QFCLOS, QFREAD, QFBKRD, QFRBA, QFREW

These routines allow quick record-by-record read of a sequential file. Disk reads are done by UIO request, several blocks at a time. QFREAD deblocks and decompresses the logical records itself, thus avoiding the overhead of SVC and MFIO processing for each logical record. The file's record format must not be U.

QFBKRD reads the previous logical record. Currently, only supported for RECFM=FC,VC files.

*Note:* These routines are re-entrant (read-only)

## QFOPEN - Open a file and define buffer area

QFOPEN opens the file (by name or ddname, etc.). The caller provides a work area, where the routines store control information and UIO buffers. The same work area must be used in all subsequent calls for that file, until the file is closed by QFCLOS. The caller should not change the contents of the work area between calls. Each concurrently open file needs its own work area.

**Calling Sequence:** CALL QFOPEN(retcod,wkarea,wklen,shortname,options,
phys,info,eofp,tag,uinf,xinf)

**Calling Sequence:** CALL QFOPEN(retcod,wkarea,wklen,-1,longname,options,
phys,info,eofp,tag,uinf,xinf)

The "options" argument is optional (can be omitted) unless phys, etc. are present. phys,info,... are optional and are the areas to be used in place of the common block areas /QFPHYS/, etc. All these arguments must be present if the routine is part of a re-entrant module, to avoid storing into the common blocks (which are part of the module). The arguments are described in detail below.

## QFCLOS - Close a file and realease buffer area

**Calling Sequence:** CALL QFCLOS(retcod,wkarea)

## QFREAD - Read next logical record

**Calling Sequence:** CALL QFREAD(retcod,wkarea,recbuf,lenbuf)

## QFBKRD - Read previous logical record

**Calling Sequence:** CALL QFBKRD(retcod,wkarea,recbuf,lenbuf)

Currently only supports RECFM=FC,VC files. Give EOF (RC=1) if attempt to read backwards too far. You can use the QFRBA to set a very high number for the RBA and the next call to QFBKRD will read the last record in the file.

## QFRBA - Set RBA (Relative Byte Address) for next read

**Calling Sequence:** CALL QFRBA(retcod,wkarea,rba)

## QFREW - Rewind the file

**Calling Sequence:** CALL QFREW(retcod,wkarea)

Equivalent to call QFRBA(retcod,wkarea,512 or 514).

## Arguments

retcod — MFIO return code. These routines may set return code 19 ("invalid arguments in call to service subroutine") for invalid calls, e.g. work area too small or not initialized by open.

wkarea — The work area for the file. See QFWRK DSECT. Contains control information followed by 1 or more 512-byte buffers. Must be on a fullword or doubleword boundary. Minimum size is about 700 bytes. Recommended size (for maximum speed) is about 10k+200. Maximum size is (2**24)-1. A very big work area should be avoided, unless the entire file will fit into the buffer area, because each I/O tries to read the greatest number of buffers possible and this could cause excessive I/O for random access in a big file.

wklen — Length of the work area, in bytes.

shortname
or
-1,longname — File name (or ddname, etc.) for the open. Maximum length is 22 for a short name and 64 for a long name. If an actual file name is provided and it is shorter than the maximum, it must be followed by a blank.

If the "LU" option is used (see below), the file name is actually a 4-byte logical unit number.

Special provision for using a file that is already open: If the options argument contains the keyword "IU", then the name argument is a 4-byte internal unit number (IU), and the file is assumed to be already open. Other option keywords are ignored if present. At the time of the call to QFOPEN, common blocks /QFINFO/ and /QFEIOP/ (or their replacements) must contain data as set by the original open. For the IU option, QFOPEN does not do an open, so it does not fill in any common blocks.

options — (Optional) a character string of the form 'keyword keyword ... keyword.' The keywords are separated by blanks and the last one is followed by a period. String '.' specifies no option keywords. The keywords specify MFIO options for the open:

DDNAME
DDORDS
MEMBER
LU
NODATE
ENQEXCL
PGMP
IU (see above)

OPEN, OKOLD, RDOK
(these are allowed but ignored; this gives compatibility with MFACT) In all cases, the open uses MFIO options okold and rdok.

phys,info,eofp,tag,uinf,xinf
These optional arguments specify replacement areas for the common blocks /QFPHYS/,/QFINFO/,etc. The phys argument must be 8 bytes long, the info argument 20

bytes long, etc. When an argument is present, it is used instead of the corresponding common block area. This is useful for cobol programs (which cannot access common blocks) and for a re-entrant module (where the common blocks are part of the module and must not be used).

recbuf — Buffer to receive the logical record. Truncation or blank padding occurs if needed.

lenbuf — The length of the caller logical record buffer (recbuf). it may be any length, including zero.

rba — The MFIO RBA (Relative Byte Address) of the next record to be read. use of qfrba is optional.

## Common Blocks Defined:

| | |
|---|---|
| /QFINFO/ | (20 bytes) infout from the open. |
| /QFTAG/ | (64 bytes) tag from the open. |
| /QFUINF/ | (46 bytes) uinfo from the open. |
| /QFXINF/ | (40 bytes) xinfo from the open. |
| /QFEOFP/ | (10 bytes) eofpt from the open. |
| /QFPHYS/ | (8 bytes) phys set after each call to QFREAD. |

*Note:* If QFOPEN provides a replacement area for a common block, that area is used instead, and the common block is not modified.

# ITS Subroutines

MUSIC's Indexed Text Search (ITS) facility provides an efficient method of locating words in large documents. The ITSRET utility provides a way to locate and display the results of a search. The subroutines described in this section provide an alternate to ITSRET. The subroutines allow you to write programs that call the search routines and check the results directly.

These routines require a work area. This work area can be obtained dynamically by a call to ITSFID or a work area can be supplied by a call to ITSFIW.

The search routines require access to an index data set built by the ITSBLD or ITSBLD2 utility. The calling sequences allow for multiple index and text data sets, however, currently only one "search set" is supported.

## ITSFID

This subroutine initializes using dynamic (Getmained) storage.

**Calling Sequence:** CALL ITSFID(rc,mhits,nss)

**Arguments:**

rc 4 bytes are returned:
=0 if ok
=1 if not enough storage available
=2 number of search sets is invalid

mhits (4 bytes) is the maximum number of hits to handle. This routine does a Getmain for the work area. Intermediate results also use this area so make it quite large. Typically 2,000 or 4,000. Each item needs 24 bytes. So MHITS=2,000 needs 48,000 bytes.

nss is the maximum number of search sets (4 bytes). Should be = 1.

## ITSFIW

This subroutine initializes using caller's work area.

Approximate amount needed is about 6,000 for fixed work space that supports 1 search set. Add to this 24 bytes per hit wanted. The fixed work space may increase in future to 10,000 or more.

**Calling Sequence:** CALL ITSFIW(rc,lhits,nss,work,lwork,mhits)

**Arguments:**

rc 4 bytes are returned:
=0 if ok
=1 if not enough storage available
=2 number of search sets is invalid
=3 Workarea not on double word boundary.

lhits is the limit to number of hits to handle (4 bytes). May not be able to do this amount due to size of input work area. If number set to -1, then take as many as will fit in area.

nss is the maximum number of search sets (4 bytes). Should be = 1.

work is the input work area location (variable length). Must be on double word boundary.

lwork is the length of work area in bytes (4 bytes).

mhits 4 bytes are returned for the number of hits that will be done. It will be the maximum number that fits in area or number as limited by *lhits.*

## ITSFOP

This subroutine opens a search set.

**Calling Sequence:** CALL ITSFOP(rc,ss,fnidx,fntext,debidx,debtext,endmark,endml)

**Arguments:**

rc 4 bytes are returned:
=0 If ok
=1 If Search Set number invalid or in use.
=2 If Index data set not created by right version of ITSBLD.
=3 Problem opening index component.
MFIO error code 256 added to return code.
=4 Problem opening text component.
MFIO error code 256 added to return code.
=5 Problem reading index component.
MFIO error code 256 added to return code.

ss is the search set number (4 bytes). Must be = 1.

fnidx is the 64 character file name of the index, or is the 1 word *deb* number (format used if already opened).

fntext is the 64 character file name of the text, or is the 1 word *deb* number. Format used if:
a) file already open by caller
b) =0 if text file will not be used by these subroutines.

debidx is the 1 word *deb* number of the index file returned after open.

debtext is the 1 word *deb* number of the text file returned after open.

endmark is the 8 character item ending string used on this search set (returned).

endml is the 4 byte count of the ending marker (returned).

## ITSFSS

This subroutine searches for a character string.

**Calling Sequence:** CALL ITSFSS(rc,sstr,lsstr,nhits)

**Arguments:**

rc 4 bytes are returned:
=0 OK, something found.
=1 No hits found. There is further information in 3rd byte of RC.

1 Only stop words looked for
2 Hits found but after boolean logic nothing left
3 Not even intermediate hits found

=2 Too many items found. Some hits missing.

=3 Problems with search string. There is further information in 3rd byte of RC.
1 Search string all blanks
2 Search word greater than maximum length (of 64)
3 Search string has quotes, periods
or other special chars around them.
4 Syntax errors

=4 Input search string length too long.

=5 Problem reading index component. MFIO error code 256 added to return code.

sstr is the search string.

lsstr is the length of search string (maximum is 256).

nhits is the number of hits found (returned).

## ITSFOR

This subroutine re-orders the matches of the last search. resulting list.

**Calling Sequence:** CALL ITSFOR(rc,so)

**Arguments:**

rc 4 bytes are returned:
=0 if ok
=1 invalid sort order request

so is the sort order combination of the file order and weight order wanted. Add the following options to give the sort order.
0 order in file (default if ORDER was not called after last search).
1 reverse order of file (last found item will be first).
2 weighted order (highest weight first).
4 reverse weighted order (highest weight last).

## ITSFRE

This subroutine retrieves results.

**Calling Sequence:** CALL ITSFRE(rc,hn,buff,nhits,layout,txtl,nent)

**Arguments:**

rc 4 bytes are returned:
=0 OK
=1 Starting hit number too large.
=1 I/O error reading from text component.
MFIO error code 256 added to return code.
(Can also include end-of-file.)
=2 Problem opening text component.
(Delayed open option used.)

MFIO error code 256 added to return code.

hn is the hit number to start with. First one is number 1.

buff is the buffer to return results.

nhits is the number of hits wanted.

layout is the layout of each return item, made up of the following optional fields. Each field will be length 4 if selected and will be in the following order:

Search set number
RBA
Weight
One line of text from text file
Length of text line

Calculate a number made up of the sum of the following wanted.

1 Return search set number
2 Return RBA
4 Return weight
8 Return length of text string. This length is obtained from the file system and may be greater than the number in *txtl*. This option should not be selected if *txtl=0*.

txtl is the maximum length of text from the text file to be read. If 0, nothing will be read. If non-zero, then use this length to read 1 logical record from the text file starting at the RBA. If the record is less than this length pad with blanks only if return length field is not given.

nent is the number of entries returned.

## ITSFCL

This subroutine closes the search.

If close all request, then storage is freed and a call to ITSFID or ITSFIW will be required if more searching wanted.

**Calling Sequence:** CALL ITSFCL(rc,ss)

**Arguments:**

rc 4 bytes are returned:
=0 if ok

ss is the search set number to close or =0 for all. If search set number given as negative number then just remove search set from list but do not close it.

## Sample Program - ITS Subroutines

```
/LOAD VSFORT
C  SAMPLE PROGRAM SHOWING THE USE OF SOME OF THE ITS SUBROUTINES

C  (REAL PROGRAMS SHOULD CHECK RETURN CODES AFTER ALL THE SUBROUTINE
```

```
C   CALLS)

      IMPLICIT INTEGER (A-Z)
C  VARIABLES TO HOLD FILE NAMES
      CHARACTER *64 FNIDX,FNDAT
C  VARIABLE TO HOLD ENDING STRING
      CHARACTER *8 ENDM
C  VARIABLE TO HOLD SEARCH STRING
      CHARACTER *80 SSTR
C  VARIABLE TO HOLD TEXT LINE FROM FILE
      CHARACTER * 80 BUFFER

C -- INITIALIZE USING DYNAMIC STORAGE AREA AND ALLOW FOR 2000 HITS.
      CALL ITSFID(RC,2000,1)
      WRITE(6,100)MHITS
100   FORMAT(' INITIALIATION CALL.  MHITS=',I6)

C -- OPEN THE SEARCH SET
C  SPECIFY FILE NAMES FOR WORD INDEX AND DATA COMPONENT
      FNIDX='$MAN:UR.WIDX'
      FNDAT='$MAN:UR.TOTAL'
      FLGS=0
      CALL ITSFOP(RC,1,FNIDX,FNDAT,DEBIDX,DEBDAT,FLGS,ENDM,ENDML)
      WRITE(6,110)DEBIDX, DEBDAT,ENDM,ENDML
110   FORMAT(' OPEN SS.  DEBNUM=',2I4,' ENDMARK=',A,' L=',I2)

C -- DO A SEARCH.  (SEARCH STRING GIVEN IN VARIABLE SSTR)
      SSTR='SENDFILE'
      LSSTR=80
      CALL ITSFSS(RC,SSTR,LSSTR,NHITS)
      WRITE(6,120)NHITS
120   FORMAT(' SEARCH FOUND', I5, ' HITS')

C -- RETURN THE FIRST LINE OF EACH SECTION THAT HAD A MATCH.
C    (TO READ MORE OF SECTION, ASK FOR RBA AND USE QFREAD OR MFIO
C     SUBROUTINES.)

      DO 40 HN=1,NHITS
      LAYOUT=0
      CALL ITSFRE(RC,HN,BUFFER,1,LAYOUT,80,NENT)
      WRITE(6,140)BUFFER
140   FORMAT(1X,1A80)
40    CONTINUE

      CALL EXIT
      END
```

# 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.

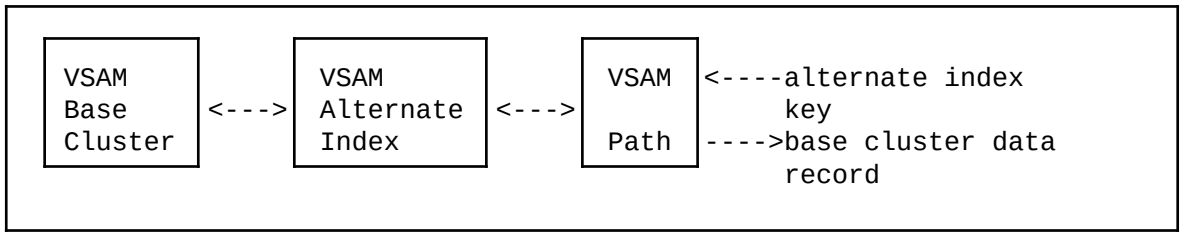
A key sequenced cluster has a field designated as the *primary key*. This key field must contain unique values; otherwise, VSAM will indicate an error condition. VSAM attempts to keep the data stored in the key sequenced cluster in the order of this primary key. Due to the patterns of additions and deletions, 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*

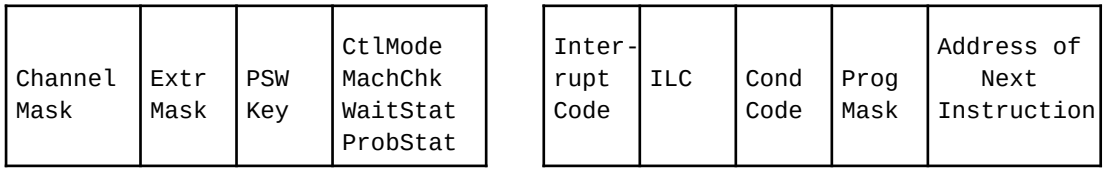
## COMMAND: Command Line

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).

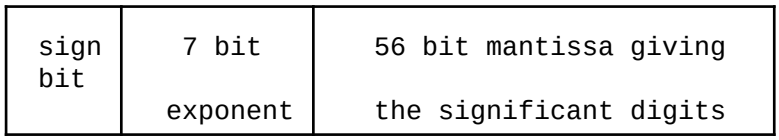
## Format of Integer Numbers

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:

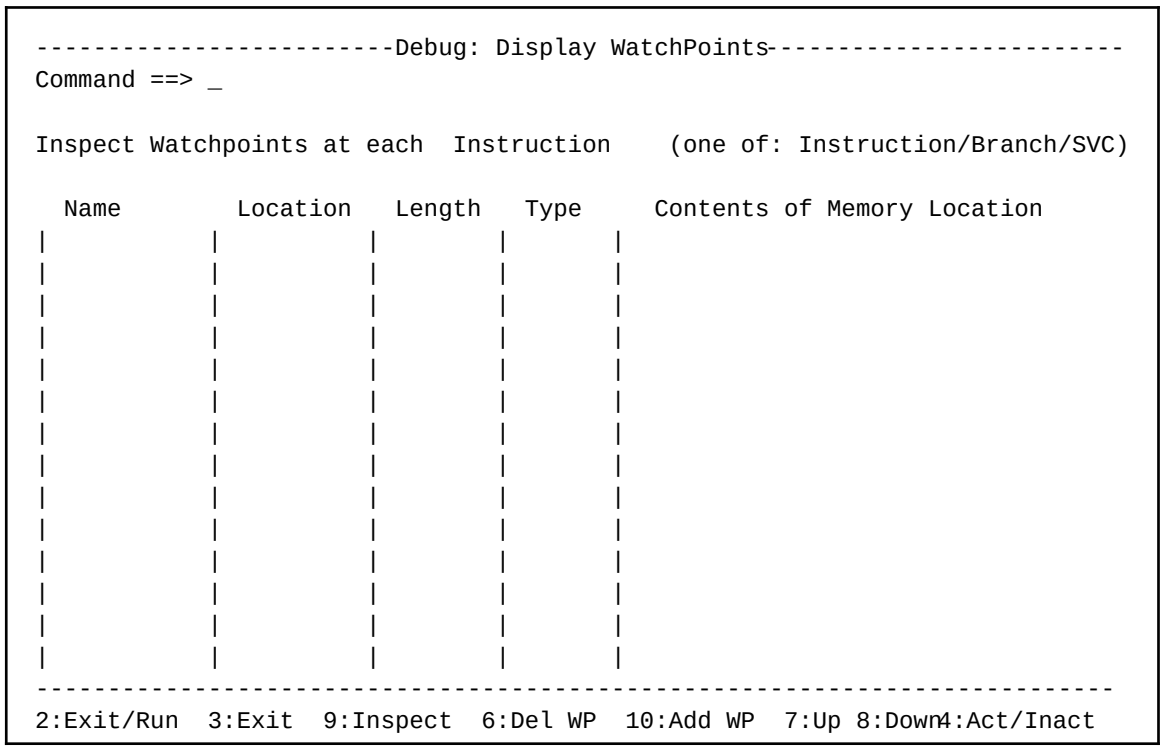



*Figure 10.4 - Displaying Watchpoints*

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

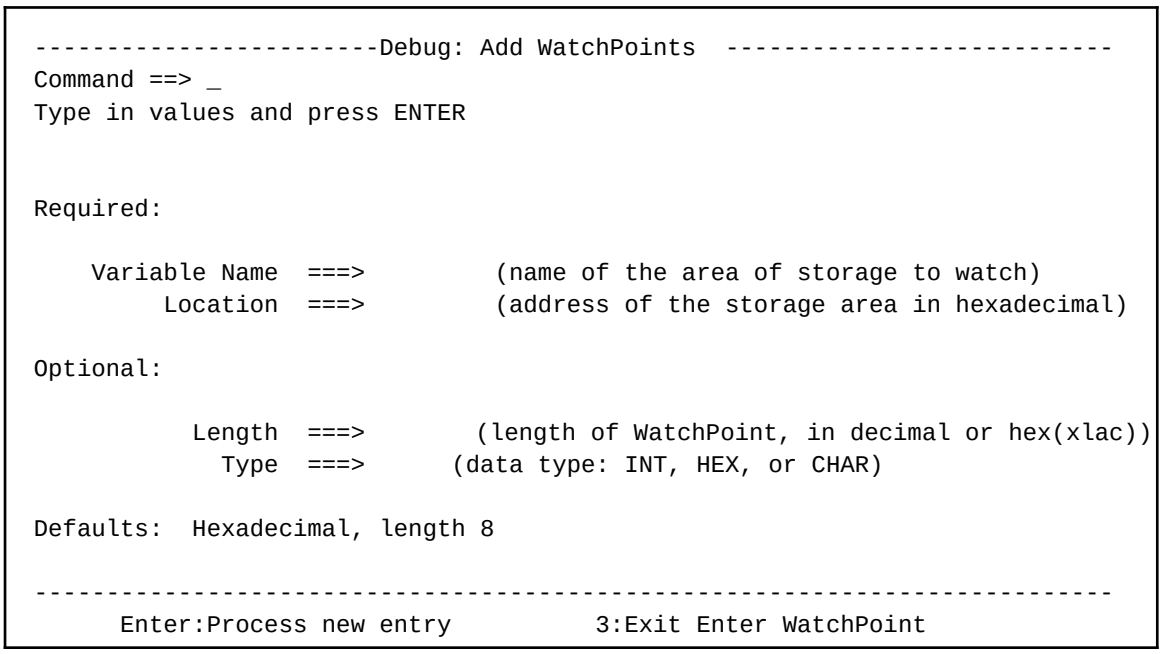



*Figure 10.5 - Adding Watchpoints*

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

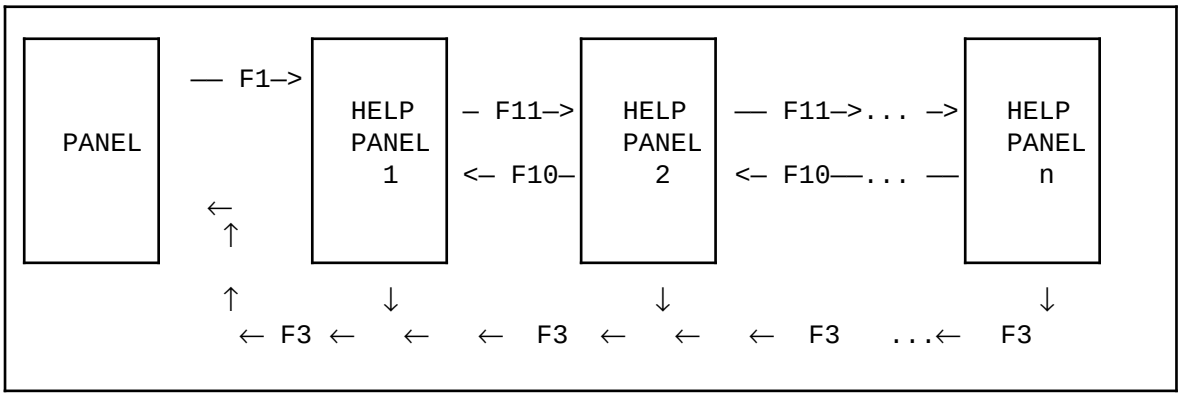



*Figure 10.7 - Flow of Panels*

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)<br>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.<br><br>The codes for the various action keys are as follows:<br><br>'01' - F1 'EN' - Enter<br>'02' - F2 'A1' - PA1<br>. 'CL' - Clear<br>. 'CS' - Cursor Select<br>'24' - F24<br><br>*Notes:*<br><br>1. If a HELP panel is available then '01' will never be returned.<br><br>2. If the automatic print panel was selected then '12' will never be returned.<br><br>3. If automatic translation of function keys was selected then codes '13' through '24' will never be returned. |
| 3-6 | Cursor Position<br>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).<br><br>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.<br><br>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.<br><br>Panel Field Number:<br>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<br>Normally, this byte is set to a blank. The following codes are available for this field:<br>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.<br>G If the Function field is set to G, then a read is performed *without* displaying the panel.<br>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.<br>R Code R reads data from asynchronous read request.<br><br>*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.<br>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<br>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<br>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).<br><br>**Examples:**<br><br>`z 10 30` the located string must occur within columns 10-30.<br>`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