VSE/Interactive Computing and Control Facility

Terminal User's Guide

Program Number 5666-302

Order Number SC33-6204-2
File No. S370/4300-39

Third Edition (June 1985)

This edition is a revision of SC33-6204-1. Changes or additions to the text and illustrations are indicated by a vertical bar to the left of the change. This edition applies to Version 2 Release 1 of the IBM Virtual Storage Extended/Interactive Computing and Control Facility, Program Number 5666-302, and to all subsequent releases unless otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30XX and 4300 Processors Bibliography,* GC20-0001, for the editions that are applicable and current.

Preface

This publication describes all the facilities needed by a terminal user to work with the IBM VSE/Interactive Computing and Control Facility (VSE/ICCF for short). It is intended for application programmers who have completed the introductory manual to VSE/ICCF, the *VSE/ICCF Introduction to Interactive Programming, SC33-6202.* In addition to the information provided in publication SC33-6204-1, a new function, double-byte character support, is documented that supports equipment (such as the IBM 5550), which may not have been announced in your country.

The information in the manual is divided into chapters as shown below.

Chapter 1. Introduction
Introduces the main concepts and facilities of VSE/ICCF.

Chapter 2. Terminal Considerations
Explains how to log on to VSE/ICCF, set tabs and Program Function keys.

Chapter 3. System Commands, Procedures and Macros
Describes the format and parameters of the VSE/ICCF system commands, procedures and macros.

Chapter 4. Editor
Explains the facilities of the editor program and describes the format and parameters of the editor commands and editor macros.

Chapter 5. Job Entry Statements
Describes the VSE/ICCF job entry statements used to execute jobs in VSE/ICCF interactive and VSE batch partitions.

Chapter 6. Dump Commands
Describes the VSE/ICCF dump commands used for debugging and problem determination.

Chapter 7. Writing Procedures and Macros
Gives guidance on writing your own procedures and macros.

Chapter 8. Utility Programs
Describes the utility programs and subroutines that VSE/ICCF makes available for standardized functions.

Chapter 9. Job Entry Considerations
Describes the facilities that you need to build job streams for execution in interactive partitions. The chapter gives hints on using such things as VSE I/O units, permanent disk files and language compilers.

Appendix A. Interactive Job Restrictions

Describes the restrictions that apply to programs in interactive partitions.

Appendix B. Context Editor

Describes the differences of certain editor commands when used under the context editor as opposed to the usage under the full screen editor.

A Bibliography, Index, and Reader's Comment Form are provided at the end of the manual.

Contents

Figures

Summary of Amendments

This publication documents **Version 2 Release 1 of VSE/ICCF**. It has the same overall organization as the *VSE/ICCF Terminal User's Guide* available with VSE/ICCF 1.3.5. Following is a list of items which are new with VSE/ICCF 2.1 and which are described in this publication:

- Inclusion of members optionally at execution time, not at time of submitting.
- Retrieval of VSE/POWER reader queue entries by the terminal user.
- Inclusion of diskette data through * $$ RDR statement.
- Access of VSE/POWER queues by several terminal users.
- 'jobsuffix' parameter to access a segment in VSE/POWER output.
- Adaptation to VSE/Advanced Functions 2.1, including the new VSE librarian.
- Testing of job execution return code in procedures and macros.
- Expiration date in /FILE statement larger than 1999.
- Macros LIBRC, LIBRL, LIBRP to move data between the VSE/ICCF library file and a VSE sublibrary.
- New /OPTION option SYSIPT/SYSLOG.
- Optionally no wait at the terminal if an interactive partition is not available.
- /COUNT command to determine the size of a member.
- Extended HELP facility.
- /RETRIEV command to recall a previously entered command.
- Termination of list mode same for /DISPLAY, /LIST, and /LISTP when displaying print data.
- Improved interactive viewing of print data through /SKIP and /SHIFT commands.
- Improved /PROTECT command syntax.
- Procedure FORTRAN supporting VS FORTRAN instead of FORTRAN IV.
- The terminal control program TTF no longer supported.
- DTSWRTRD macro and DTSSCRN macro supporting 3270 Extended Data Stream features.

- Double-Byte Character Support (DBCS), if an IBM 5550 terminal with 3270 Emulation is installed.

Chapter 1. Introduction

This chapter gives an overview of the main functions and facilities of the VSE/Interactive Computing and Control Facility (VSE/ICCF). It is assumed that you are already familiar with at least Part 1 of the *VSE/ICCF Introduction to Interactive Programming*, SC33-6202. Here are some of the functions that VSE/ICCF makes available to you:

- Enter programs and data and save them in a library.
- Display and edit these programs, compile, run and debug them.
- Write programs which interact with the terminal and which use the screen of display terminals.
- Build streams of interrelated job steps and save them in libraries.
- Build procedures or macros including variable parameters and run them, or save them in a library.
- Define procedures with logic and loop control facilities.
- Submit VSE/ICCF or VSE JCL job streams to VSE/POWER to be run in another VSE partition, and display the print output.
- Submit jobs via VSE/POWER to be run at another node, or direct output from jobs to another node, and receive job ending messages for both local and remote jobs.
- Save print output from interactive partitions in a library member and later display it in print format.
- Communicate with the system operator, with the VSE/ICCF administrator, or with other terminal users via messages.

The HELP Facility

When you are working under VSE/ICCF, you can type HELP on your terminal and press ENTER. You will then receive a display of information concerning VSE/ICCF commands, macros, procedures, and job entry statements. For information on setting up your terminal and using VSE/ICCF, see also *Chapter 2. Terminal Considerations.*

Modes of Operation

During a session your terminal is always in one of six modes of operation. The current mode is shown in the scale line of your screen. These modes determine in part what commands you can enter into the system at any given time. For example, the /INPUT command, which places the terminal in input mode, can only be entered when your terminal is in command mode. On the other hand, the /END command can be entered in input mode but is invalid if entered when your terminal is in command mode. Here is a summary of the six modes of operation and the command abbreviations as they appear in the scale line of your screen.

- Command mode (CM) is the basic mode when the system is not in message, edit, list, input or execution mode. Command mode is usually indicated by the *READY message.

- Input mode (IN) is set by the /INPUT command. After the /INPUT command has been entered, you can type data into your input area. To exit from input mode and return to command mode, enter the /CANCEL, /SAVE, /ENDRUN or /END command.

- Full screen edit mode (FS), or simply *edit mode*, is set after you called the editor via the ED macro. When you are in edit mode, you can directly modify your data anywhere on the screen. In addition, you can use 'line commands' to add, delete, duplicate etc. lines in your file. From edit mode, you can place your terminal into full screen editor input mode ('editor input mode' for short) by issuing the INPUT command. You leave edit mode by issuing the QUIT, END or FILE command.

  VSE/ICCF has another editor mode: context edit mode (ED). This mode is set after you called the context editor (see Appendix B) by issuing the /ED command. In context edit mode, you locate your data by searching for a character string. You can modify only one line of data at a time. As in full screen editor mode, you can place your terminal in context editor input mode by issuing the INPUT command.

  In this publication, only when necessary will the prefixes 'full screen' and 'context' be set in front of the terms edit mode and editor input mode.

- Execution mode (EX) is entered whenever a program execution is in progress or queued for execution. Execution mode remains in effect until the background job ends and completes printing or until you issue the /CANCEL command. While still in EX mode, you can do editing and other work (see also "Asynchronous Execution Mode" on page 1-4). While print output from the executing program is being spooled to the terminal, 'SP' is displayed. If the program in execution requests input from the terminal, 'RD' is displayed.

- List mode (LS) is in effect while a display is in progress. For example, if a /LIST command was issued for a member and the displayed data could not be contained completely on one screen, your terminal is placed into list mode. Each time you press ENTER, the next screen will be displayed.

- Message mode (MS) is in effect while messages are being displayed.

The Command Language

VSE/ICCF has the following command groups:

- System Commands - System commands direct general system functions; for example, the /LOGON and /LOGOFF commands begin and end a session. The /INSERT command inserts library members into your input area, or into another library member. The /SAVE command instructs the system to save the contents of the input area in the library. You can also remove members from a library with the /PURGE command or, using the /RUN command, cause the job in your input area to be run. With the /SEND command you are able to send messages to the system operator, or to other terminal users.

- (Full Screen) Editor Commands - The term 'full screen' refers to the fact that nearly the entire screen is available for entering data or commands.

  Some of the full screen editor commands, such as VIEW, SCREEN and FORMAT, control what is displayed on the screen and how it is displayed. Other commands, for example NEXT, LOCATE, or CHANGE, relate to moving the line pointer, locating an area within the file or changing or adding data within the current line.

  Multiple editor commands can be entered in the command area by separating individual commands with logical end-of-line characters.

  The editor **line** commands are a subgroup of the editor commands. They manipulate lines as a whole: add, delete, move or copy single or multiple lines. You enter these commands in the *line command area* of the affected line(s).

- Context Editor Commands - The context editor commands are identical to the editor commands described above. Under the context editor, some of these commands have a slightly different function. These differences are described in Appendix B, "Context Editor" on page B-1.

  Context editor commands are required either when you work with a typewriter terminal or when you use editor functions in macros and procedures or in the DTSBATCH utility.

- Job Entry Statements - Job entry statements direct and control the execution of jobs. They look very much like system commands except that a system command is put into effect immediately, whereas job entry statements are placed in job streams and the functions that they request are not carried out until the job is actually run. Some examples of these statements are: the /LOAD statement defines the start of a job and indicates which compiler or program phase is to be loaded; /INCLUDE instructs the system to include specified members in the job stream, and the /OPTION statement sets various job processing options.

- Dump Commands - The dump commands enable you to display information from programs that have ended abnormally, provided you have specified the DUMP option in your /OPTION statement. For example, the DISPLAY command shows the contents of program storage and the general or floating-point registers. The LOCATE command is used to locate characters within storage. The POINT command sets the scan locate pointer to a specified address, and the STATUS command obtains various status displays.

- Procedural and Macro Commands - VSE/ICCF includes some procedures and macros that provide you with additional programming facilities. A procedure or macro is a library member that contains executable statements and commands and/or data. A procedure is invoked by specifying a file name as a command, for example SUBMIT. The same is also true for macros -- but with one exception. Macros invoked in edit mode must be specified with the prefix '@', for example: @COPY. In both cases, all statements and commands contained in the specified file are then executed. The usage of each of the IBM-supplied procedures and macros is described in Chapters 3 and 4 in alphabetical order. The supplied procedures and macros can also be used as examples for writing your own procedures and macros. For more information on writing your own procedures and macros see Chapter 7, "Writing Procedures and Macros" on page 7-1.

Foreground Versus Background

All VSE/ICCF system and context editor commands are executed in what is called a foreground task, whereas all compilations and executions are performed as a background task.

The commands that are processed in the foreground have a higher priority than the functions performed in the background. This is to maintain an adequate level of response to terminal users who are doing command work, which requires very little of the system's computing resources. The number of execution requests that can run concurrently is decided at your installation when VSE/ICCF is being tailored. These execution requests are initiated whenever you issue the /RUN or /EXEC commands.

Whenever a /RUN or /EXEC command is issued, the job is queued for execution. If a block of virtual storage (called an interactive partition) is available in which to run the job, the job is scheduled and, thus, becomes a background job. You can have only one execution request running in the background at a time, during which time you can only carry on with foreground command work.

Interactive partitions are associated with up to four classes, which allow you to direct your jobs to partitions that have the characteristics that you need for a particular job, for example, pre-allocated work files, larger size, and so on.

The terms 'foreground' and 'background' relate only to the VSE/ICCF environment and should not be confused with the terms 'foreground' and 'background' as they apply to VSE, or to the submitting of jobs for batch execution via VSE/POWER.

Asynchronous Execution Mode

When you have initiated an interactive partition execution, you can issue the /ASYNCH command which causes asynchronous execution mode to be set and command mode to be entered. In this mode you can perform functions such as editing while your execution is in progress. Entering the /SYNCH command will resynchronize your terminal with your execution in the background.

While in asynchronous execution mode, you cannot issue the /INPUT command. However, if you want to enter data while the execution is in progress, the data can be placed in a dummy library member using the editor. Also, the input area cannot be updated or edited while an asynchronous execution is in progress.

The VSE/ICCF Library File

As a user of VSE/ICCF you have access to one or more libraries, which you can own exclusively or share with other users. Each library has a directory which can contain any number of entries (however, the installation may have, for reasons of efficiency and space management, placed a limitation on the maximum number of entries in your directory).

Each directory entry contains a name which you have given to your data. A single file of data in the library is called a 'member'. Each member is represented by a member name and location in the directory. (For reasons of efficiency and space management, it is to your advantage to keep the number of members in your library as small as possible. Use the /PURGE command to remove entries from the library which are no longer needed.)

Each member in the library consists of one or more 80-character records. The records can be programs, data, object decks, procedures, VSE/ICCF job streams, documentation, VSE JCL or any combination of these. Your installation may have placed a limit on the number of statements which can be contained in any one of your members. This limitation was set for reasons of efficiency. However, if you must exceed this limitation, simply start another member.

It is possible to logically connect more than one member by using the /INCLUDE job entry statement. To VSE/ICCF, there is no difference between having all of the data in one member or in several members. In fact, it is more efficient for you to update multiple small members rather than one large member. Each member (file containing records) that you save in your library must be identified with a unique name. The name must be from 1 to 8 characters in length and the first character must be alphabetic.

You can transfer library members from the VSE/ICCF library file to a sublibrary of your VSE system by invoking the LIBRC macro. By using the LIBRL and LIBRP macro you can copy a member from a VSE sublibrary to a VSE/ICCF library.

Types of User Library

Basically there are three types of library in VSE/ICCF: PRIVATE, PUBLIC and COMMON libraries. The main difference between these three types of library is the degree of security that each type offers you, and this will be discussed in more detail presently.

For the moment, however, it is important to note that these libraries can be referred to in other ways, depending on how they are allocated to you, and how you use them. For example, as a user of VSE/ICCF you may be given access to more than one private library (specified in your user profile), in which case you must select one of these libraries as your main library and the others -- including the public libraries to which you have access -- will become your alternate libraries. In addition to your main and alternate (private and public) libraries you will also have access to the COMMON library.

Another way of looking at libraries concerns the way you use them during terminal operations. At any given time during a session you will have access to three libraries: two private or public libraries, and the common library, if you so desire. VSE/ICCF searches these libraries in a set order and not according to the type of library concerned. Thus, to be able to keep track of which of your libraries you are currently searching first, and which second, you will find it convenient to think of the first two libraries as your primary and secondary libraries, regardless of the type of library involved (the common library is always the last library to be searched). The order of your primary and secondary libraries can, of course, be changed at any time with the /SWITCH command.

Public versus Private Libraries

A public library is accessible to you via the /SWITCH or /CONNECT commands or automatically at logon time as your main library if specified in your user profile. On the other hand, a private library is only accessible to you if your user profile indicates that you have access to that library (via logon or the /SWITCH or /CONNECT commands).

Shared versus Owned Libraries

Your data has the highest level of security when you are the sole owner of one or more private libraries. That is, only you, or someone signing on with your userid, can access libraries that you own for any purpose whatsoever. Likewise, the user with an 'owned' private library only has access to common data, to the common library, to any public libraries, and to his own data. An exception to this might be if he also has access to an alternate private library (via the /SWITCH command) that is shared with other users. He cannot access another user's data even if it is flagged as PUBLIC.

A 'shared' private library on the other hand can be used by two or more users (as identified by their userids). Shared private libraries allow two or more users working on the same material to have access to each other's data so long as that data has been entered as PUBLIC. A user of a shared private library can secure his or her data by entering it as PRIVATE or by password protecting it.

The Alternate Library

Certain users can access more than one private library. If your user profile has been set to include one or more alternate libraries you can issue the /SWITCH command to switch access between such an alternate library and your own main library. In addition, the /CONNECT command can be issued to logically connect an alternate private or public library to your primary library. Members can only be updated in the current primary library, which means that commands like /EDIT, /SAVE and /PURGE will only work on members in this library.

Public versus Private Data

All data which is saved in a library is entered as public or private, depending on whether you specified public or private when you issued the /SAVE command. If you do not specify either public or private in the /SAVE command the default (public or private) as indicated in your user profile is taken.

Private data can be read by any user who shares the library unless the 'Alternate Security' option has been selected. Public data can be accessed by any user who 'shares' the library. Private data can be modified only by the user who entered it. The public versus private designations on a given member can be changed by issuing the /PROTECT command or by replacing the data using the /REPLACE command.

Generation Member Group

A library member can be designated as a generation member group with from 2 to 10 entries in the group. Then when the member is saved or replaced, the new data becomes the current member of the group and all previous members of the group except for the last member have their generation indicator increased. The last member of the group is purged so that the number of members in the group is always constant. (See the /GROUP command in *Chapter 3. System Commands, Procedures and Macros.*)

Having a generation member group gives you protection in case an error occurs. You can always use the previous generation of the member to correct the problem.

Common Data and the Common Library

There are two ways of making library members accessible to all users at all times: 'Common Data' and 'The Common Library'. Common data refers to a special type of member, of which only one copy exists in the entire library. Yet, each user has an entry for this member in his or her library directory.

A common data member usually contains common subroutines or procedures needed by all users. Common data is public in the sense that all users have access to it. However, only the VSE/ICCF administrator can update it; the individual terminal user cannot normally update common data. However, a common data member can be inserted by a user into the input area, updated, and saved in his or her library under a different name.

Rather than elect to have all common members represented in each directory (the common data approach), the installation may have chosen to implement a common library. If a common library exists within the system, all directory searches scan the common library directory if the desired member name is not found in your own primary or connected libraries.

It is also possible that both techniques have been employed in your environment. You can find common data within your own library (/LIB FULL command) and also find the presence of a common library (/LIB COMMON command).

Library Efficiency

The way in which you manage your library will influence both your own efficiency and also that of the VSE/ICCF system. Keep the following points in mind when dealing with your library:

- The fewer members there are in your library, the faster the search will be for any one member. Purge unnecessary members from your library regularly, or use multiple libraries with few members rather than a single library with many members.

- The fewer records there are in any one member, the easier and faster it will be to edit, update or copy it. For example, if a program is large but is modularized into logically related program segments, a given change may affect only one module. Thus, making a copy of this module for editing, and the editing itself, should take less time.

- Use the /INCLUDE facility to logically connect two or more library members into a single logical data file for execution.

- New members are added to the front of a directory, or as close to the front as possible, because recently saved data will probably be used more often. This saves directory look-up time. However, if one of your active entries works its way toward the bottom of the directory, you can move it closer to the top again by inserting it into the input area (/INSERT), purging it from the library (/PURGE), and resaving it (/SAVE).

- After your library member reaches a complete or inactive state, you can issue the /SQUEEZE command to physically compress it. This usually results in a 60% to 75% disk space saving. Once the library member has been compressed it cannot be edited, updated or executed. When you want to perform these functions, the member must be decompressed by inserting it into the input area (/INSERT command). However, the /LIST command allows you to view all or a portion of a compressed member without having to first decompress it.

Temporary Storage Areas

Besides having access to one or more libraries, each terminal user also has access to four storage/work areas, which are part of the VSE/ICCF library. There is an INPUT, a PUNCH, a PRINT and a LOG area.

- THE INPUT AREA can be used for data that you type in, or for data from a library, and is created when you enter the /INPUT command. All data that you later type in is placed in this area. The /INSERT command can also be used to insert all or a portion of a library member into the input area.

  Once data exists in the input area, it can be added to, modified or deleted after issuing the ED macro. The editor input mode also provides a means of typing in more data without adding an /INSERT command on every line, and without having to leave edit mode and enter input mode.

  The input area can be cleared by entering the /INPUT command followed by /CANCEL. The /CANCEL command, while in input mode, always clears all data in the input area. The contents of the input area are also changed by /EXEC and /RUN when these commands are specified with a name operand, or a procedure is invoked.

  Another way of clearing the input area is to save its contents in the library using the /SAVE, /REPLACE or the editor SAVE command.

- THE PUNCH AREA contains the punch output of jobs run in the background. The output of programs that use the VSE symbolic units SYSPCH or SYSnnn (where nnn is defined by the installation as the punch programmer unit) is directed to the punch area. For example, when a compilation is performed, the object program is placed in the punch area. From this area it can be saved in a library member or, as is normally the case, loaded into storage for execution by the LINKNGO program.

  But compilers are not the only programs which can use the punch area. You can write 80-character records to the punch area from your own programs. The records thus written can be read back in another program or saved as a member of the library.

  The contents of the punch area can be physically transferred to the input area using a special form of the /INSERT command. The contents of the punch area can also be logically included as input via a similar form of the /INCLUDE statement. Punched output can also be directed to a library member rather than to the punch area. This avoids lost time in transferring the contents of the punch area to the input area when this is to be saved as a library member.

- THE PRINT AREA (sometimes called the 'spool' area), is an intermediate storage area used to hold print output from background executions until the output has been displayed on your terminal. The default number of print lines which can be held in this area is controlled by the installation; however, you may, within limits, modify this value with the /SET command, for example to hold larger listings. A user profile parameter sets the limit to which this area can be increased.

  Similar to the punch area, the contents of the print area can also be transferred to the input area via the /INSERT command. This data can also be included logically as input using the /INCLUDE statement, and print output can be directed to a library member rather than to the print area.

- THE LOG AREA can be used to record details of your terminal activity. To specify logging for your terminal you would use the /SET LOG command. Logging causes the input/output from and to your terminal to be saved in the log area. Then, by entering the /LIST $$LOG command you can display the commands you have just entered. The most basic level of logging saves only commands with the time and mode of entry. Other logging options provide a more comprehensive record, including all input (commands and data) and/or all system responses.

Languages Supported

VSE/ICCF attempts to create a background processing environment which should allow most 'batch' (that is card-input, print output) type compilers to operate without modification as long as they conform to VSE programming standards.

Therefore, you can write and run programs in the following languages: ASSEMBLER, COBOL, BASIC, PL/I, FORTRAN and RPG.

Specifically, the VSE language compilers which can be used to compile and run programs are VS FORTRAN, DOS/VSE ASSEMBLER, DOS PL/I OPTIMIZING COMPILER, DOS/VS FULL ANS COBOL, DOS/VS RPG II and VS BASIC.

Some of these compilers are IBM program products which your installation may or may not have implemented. Therefore, you should check with the VSE/ICCF administrator for information concerning compiler and language support.

The Interactive Interface

An interactive program is one which accepts data from the terminal and displays responses based on the input data back to the terminal. You can write interactive programs in any of the languages mentioned above.

To write an interactive program, merely pretend that you are requesting information from the system console (SYSLOG) and displaying answers back on the system console. That is essentially how easy it is. When the program is run, the reads from your terminal are converted to reads from the console, and the writes to the console are converted to writes to your terminal.

If your programming language does not support interaction with the system console (for example, FORTRAN and BASIC), use the logical units SYSIPT and SYSLST for terminal input and output. A special option (INCON on the /OPTION and /DATA cards) causes any SYSIPT directed input (VS FORTRAN unit number 5 or INPUT statement in BASIC) to be requested from the terminal.

Thus, any program written to read cards and to write on a printer can be made interactive.

The LINKNGO Program

VSE/ICCF has its own link, load and go program called LINKNGO which converts the output of the language compilers (except VS BASIC and any other compilers which perform their own load) into an executable program.

Normally, you do not have to request execution of the LINKNGO program because it is assumed that you want to run your program after it is compiled. If you do not want LINKNGO invoked, the NOGO option can be used to prevent execution of the compiled program.

Occasionally, however, you may want to invoke LINKNGO explicitly (/LOAD LINKNGO). For example, a job with two compiles and executes would require a specific invocation of LINKNGO, because LINKNGO normally executes at the end of all compiles on the assumption that all compiles are to be linked together as one loadable program.

The following job would run the COBOL compiler and the assembler. Then the COBOL object module and the assembler object module (in this case a subroutine requested in the COBOL program) will be loaded together and run as a single program.

```
/LOAD FCOBOL
     .
     . (COBOL program)
     .
   CALL SUBRTN
     .
     .
/LOAD ASSEMBLY
SUBRTN START 0
     .
     . (assembler program)
     .
```

The LINKNGO program would be run once at the end of the job and the COBOL program would be combined with the assembler subroutine and run.

However, in the following example, it is desired to load and run each compile as it completes.

```
/LOAD FCOBOL
     .
     . (COBOL program)
     .
/LOAD LINKNGO (Load and run the program)
/LOAD FCOBOL
     .
     . (COBOL program)
     .
/LOAD LINKNGO (Optional - Load and run the program)
```

The second /LOAD LINKNGO was optional because LINKNGO would have been run automatically upon reaching the end of the job. Any data cards to be read could have been inserted following the /LOAD LINKNGO.

It should also be pointed out that encountering a /DATA card in the job stream following a compile will force execution of the LINKNGO program unless /OPTION NOGO had been specified.

The Include Facility

When building jobs to be run, the programs to be compiled or the data itself, or both are usually part of the job stream. However, it is often more convenient to have programs and/or data broken down into smaller, separate sub-modules, by grouping them together with successive /INCLUDE statements.

Normally you might enter a FORTRAN compile and run with the following job entry statements and 80-character records.

```
/LOAD VFORTRAN
  (FORTRAN Source Program)
/DATA
  (Input 80-character records)
```

However, if the data was in a separate member of the library named FORTDATA, the job stream might look as follows:

```
/LOAD VFORTRAN
  (FORTRAN Source Program)
/DATA
/INCLUDE FORTDATA
```

Or, the program itself may be in yet another member named FORTPROG in which case the job entry statements would look like this:

```
/LOAD VFORTRAN
/INCLUDE FORTPROG
/DATA
/INCLUDE FORTDATA
```

Now, if the FORTRAN program was large and was in three library members named FORTPR1, FORTPR2 and FORTPR3, the following job stream would result:

```
/LOAD VFORTRAN
/INCLUDE FORTPR1
/INCLUDE FORTPR2
/INCLUDE FORTPR3
/DATA
/INCLUDE FORTDATA
```

/INCLUDE requests can be nested eight levels deep. That is, it is permissible to have /INCLUDE statements within library members which are themselves the object of an /INCLUDE.

VSE/ICCF Job Streams

VSE/ICCF job streams can be built in the input area and either run with the /RUN command or saved as members of the library and run with the /EXEC command. A job stream consists of job entry statements (such as /LOAD, /FILE, /OPTION, /UPSI and /ASSGN) and can also consist of source programs and/or data. If the source programs and data to be processed within a job stream are in separate members of the library, they can be logically included in the job stream using the /INCLUDE statement.

A job stream can consist of a single program execution (single step job) or multiple program executions (multi-step job). Each step in a job stream begins with the /LOAD statement unless an execution of the LINKNGO utility is implicit, in which case a /LOAD statement is not required.

You can define existing VSE files (SAM, DAM, VSE/VSAM) in your VSE/ICCF job streams via the /FILE statement, thus eliminating the need to have all DLBL EXTENT cards present in the VSE/ICCF startup deck. In addition to being able to define VSE files via the /FILE statement, you can also define dynamically allocated files, and assign unit record devices to various 'SYS' units rather than having to use the system defaults. You can also cause a job stream to conversationally prompt the terminal for a job entry statement at execution time. Also, if errors are detected in job entry statements, the terminal is prompted for reentry of the statement in error. VSE/ICCF allows programs running in interactive partitions to set the VSE operator communication exit. The exit is activated by the /ATTEN command or by the 2741 ATTN key or the 3270 PA1 key.

The User Profile

Each user of the system has a unique, four-character user identification (userid). Associated with this userid are two 80-character records called the user profile. Some typical fields in your user profile are:

- Your userid, logon password, and the name of your logon procedure, if you have one.

- The identification of your library, including the identification of any alternate private libraries to which you have access.

- The maximum number of lines that you can enter in any given library member, or in your punch and print areas.
- Your security level and time limits for any given execution.
- Accounting information, such as number of logon requests, number of commands entered, number of execution (background) requests and execution time.
- Direct access storage (library) usage information.
- Privileged versus non-privileged status information.
- Whether prompting by line number is the default.
- Whether you can enter jobs into background execution or whether you are limited to submitting to 'batch' for compilation and execution.
- Whether your data is entered into your library as PUBLIC or PRIVATE if not explicitly stated.

The user profile thus contains the information that informs the system about your requirements and security level.

Access Control

VSE/ICCF has several different means to protect the VSE/ICCF libraries and data files, and in particular to protect phases from unauthorized execution in interactive partitions. Phases and data files are protected by VSE instead of by VSE/ICCF if your system is IPLed with the SEC=YES option in the IPL command SYS, and all jobs submitted for execution in VSE partitions must be 'logged on' by a job control ID statement. Access control under VSE/ICCF is implemented as follows:

- User identification - Each VSE/ICCF user has a four-character user identification (userid), which must be specified when logging on to the system. This userid also identifies the owner of members in a library that is shared by other users.
- Logon Password - A three to six character password must be specified at logon time. This password should be altered from time to time by the VSE/ICCF administrator to ensure continued protection. In certain cases, you may be given the right to alter your own password via a system command.
- Shared versus Owned Libraries - The highest level of data protection is attained when you own a library, or libraries. Access to an owned library, for any purpose whatever, is only possible via the owner userid. A shared library, on the other hand, is accessible to a number of users, as identified by their userids. Data in a shared library is accessible to other users of the library, provided the data has been entered as public. To secure data in a shared library, you must enter it as private, or protect it with a password.
- Alternate Security - If the VSE/ICCF administrator has implemented the 'Alternate Security' option, the right to access public and private data is determined differently. With normal (default) protection, public data can be read or written by any one (who shares the library) while private data can be read by any user of the library but only updated by the user who entered the data. Under the alternate security approach, public data can be read by any user but only updated by the user who entered the data, while private data cannot be accessed for any purpose except by the user who entered the data.

- Member Passwords - Any user can save members in the library with a four-character password. The password must not be PRIV or PUBL and must begin with an alphabetic character. Any later reference to this member name must include the password as a suffix. For example, /LIST FORTPROG PASS would display the member FORTPROG which was protected with the password PASS.

  If a password protected member is flagged as private, only the user entering the data (or someone else using the same userid and who also knows the password) can modify the data. If a password protected member is flagged as public, any user who has access to the library and who knows the password can modify the data.

  The password of a member in the library can be added, changed or deleted using the /PROTECT command. The use of a password reserves all forms of access (read and modify) for the user who entered the member.

- Controlled access to user files and programs - VSE/ICCF allows you to define files and programs to which only authorized users may have access. If the VSE access control function of VSE/Advanced Functions is active (SEC=YES specified in the IPL command SYS), the VSE/ICCF protection facilities are replaced by system-wide access control facilities that offer control for batch and/or interactive applications. These facilities can also be extended using the VSE/Access Control - Logging and Reporting program. More information on these facilities is available in *VSE/Access Control-Logging and Reporting, General Information*, GH12-5130 and *VSE/Advanced Functions System Management Guide*, SC33-6191.

Dynamic Space Allocation

The VSE/ICCF dynamic space allocation feature allows you to allocate disk work file space when you are scheduling a job for execution. You thus do not have to pre-allocate this space at VSE/ICCF initiation time. Dynamically allocated files can be specified for sequential and direct access files as permanent or temporary. That is they can be retained from day to day, or they may be accessible only while the job, or job step, that allocates them is running. The retention of these files is subject to installation procedures. For example, if all dynamic space areas are 'cold-started' each day, no user files can be retained.

Job Execution Under VSE/POWER

Besides enabling you to run jobs in interactive partitions, VSE/ICCF also allows you to run VSE/ICCF and VSE jobs in VSE partitions. VSE/ICCF must, however, be running in a system that includes VSE/POWER. These jobs are submitted to the VSE/POWER reader queue just as you would present normal VSE jobs for processing via VSE/POWER.

The print output from such a job can be disposed of in several ways. It can be routed to your terminal, into your print area, or into a library member. Or it can be routed to a printer associated with your terminal, to the system printer, or to a VSE/POWER RJE printer. Punch output can be placed into a library member or into your punch area. If a password is added to the SUBMIT procedure, or to the VSE/POWER JOB, LST or PUN statements for jobs submitted to VSE/POWER, the output from these jobs is protected against unauthorized retrieval. The status of these jobs can be requested at any time during execution.

If your installation is part of a VSE/POWER controlled network (PNET), you can transmit job streams to other nodes. You can control the execution of these jobs and return the results to your terminal. You can also exchange messages with users at other VSE/POWER controlled nodes of the network.

The following commands, procedures and macros help you to handle VSE/POWER jobs:

SUBMIT procedure	to submit jobs to VSE/POWER
RELIST macro	to transfer the contents of the print area or a library member to the printer
/DQ command	to display VSE/POWER queues
/LISTP command	to display output data from the VSE/POWER LST queue
/LOCP command	to locate a character string in the displayed VSE/POWER list output
/CTLP command	to issue a VSE/POWER command (VSE/ICCF administrator)
/ERASEP command	to erase a VSE/POWER queue element
/ROUTEP command	to route a VSE/POWER queue element
/STATUSP command	to display the status of a VSE/POWER job
/SKIP command	to skip forward and backward in a VSE/POWER print file (in a /LISTP operation)
GETL procedure	to retrieve VSE/POWER list queue output
GETP procedure	to retrieve VSE/POWER punch queue output
GETR procedure	to retrieve a VSE/POWER reader queue entry
/MSG command	to review the VSE/POWER messages sent for you
/SET MSGAUTO ON	to get the incoming VSE/POWER messages displayed automatically

Message Transmission Facilities

VSE/ICCF terminal users, including the VSE/ICCF administrator and the system operator, can communicate with each other in three ways:

- The **BROADCAST** facility allows the VSE/ICCF administrator to send information to all terminal users in the form of messages, which are displayed when the user logs on. These messages are usually short and of general interest to all users, for example:

```
*   VSE/ICCF 2.1.0 INSTALLED  23/09/85
*   VSE/ICCF WILL BE UP UNTIL  7:00 PM
*   SEE MAIL FOR NEW SUBMIT VSE/POWER JCL DEFAULTS
```

- The **MAIL** facility can be used by the VSE/ICCF administrator to make larger amounts of information available to users of the system. This information is displayed only on request. An example would be descriptions of installation dependent items, like VSE/POWER defaults during a SUBMIT.

- The **NOTIFICATION** facility is the usual means by which terminal users and the system operator communicate with each other. All parties can send and receive messages, which are typically only of short-term interest. They are displayed according to the option set in the receiver's user profile (either automatically or on request via the /MSG command), and are then deleted. Some examples are:

```
- 13:20 MSG FROM AAAA PLEASE SEE ME BEFORE YOU LEAVE
- 14:00 MSG FROM COPER PLEASE LOGOFF
```

Message Transmission Facility	FROM		TO		Life of Message
	Originator	Commands	Receiver	Commands	
BROADCAST	VSE/ICCF administ- rator	ADD BROADCAST*	All termi- nal users	During LOGON	Kept until replaced manually
MAIL	VSE/ICCF administ- rator	* ADD MEMBER (I,$MAIL) /EDIT A$MAIL	Terminal users (all or specif.)	/MAIL	Kept until deleted manually
NOTIFICATION	Terminal users, system operator	/SEND	Terminal users (all, specif.), or system operator	During LOGON /MSG or automa- tic (see /SET MSGAUTO)	Deleted automatic- ally after being displayed
* DTSUTIL commands					

Figure 1-1. Message Transmission Facilities

The Editors

VSE/ICCF contains two editing facilities: the context editor, which can be used with any terminal, and the full screen editor, which can only be used with the IBM 3270 Information Display System.

The Context Editor

The context editor is a general purpose text editing program that allows you to create and modify files from the terminal. A file can be either a member of your library or data in the input area.

The context editor consists of a group of commands and macros which locate and modify strings of information within a file, move a logical pointer from one record to another, perform other control functions such as setting logical tabs or the string scanning zone, replace or insert lines and edit files by line number.

All context editor commands have an immediate effect upon the data being edited, that is, the change is made to the actual file when it is requested. If you are using a screen terminal, your position within the member being edited is indicated by a scale line across the middle of the screen. The current line is the line directly below the scale line. You select the number of lines (1-20) that you want to appear before the current line and the number of lines (1-20) that you want to appear after it. These parameters are selectable during editing. All lines below the scale line appear in heightened intensity. You can turn full screen verification on or off. 'ON' is assumed for local 3270's (see the VERIFY command in Appendix B).

By placing sequence numbers within a member, you can use line number editing as well as context editing. Finding a given line in line number editing requires simply typing in the line number. To replace a current line or to add a new line, type in the line number followed by the data: the editor will automatically position you to the correct point in the file and make the desired change. Line number editing and normal context editing can be used on the same member concurrently (see the LINEMODE and 'nn' commands in Chapter 4, and also some hints in Appendix B).

For editing large members, you can create an index for the member, which enables you to move the line pointer quickly. When line number editing is in effect with indexing, finding specific line number positions in the member is also done using the index. Moving and copying data from one part of a member to another are carried out by the @MOVE and @COPY macros. For example, '@COPY 10 LOC /IDEA/' would cause the ten lines at the current position to be copied behind the first line where the string 'IDEA' occurred.

The Full Screen Editor

The full screen editor was designed to work with the IBM 3270 Information Display System. It thus provides you with all the features that these terminals offer, such as user defined PF key options, and minimum data transmission (changed data only). Besides including all the functions of the context editor, the full screen editor also allows you to change and insert data anywhere on the screen simply by positioning the cursor and making the desired change. The split screen facility allows you to display and edit up to eight files, or different areas of the same file, on the screen at one time. You can move or copy data within the same file, or between files, either on the present screen or onto subsequent screens. If extra room is needed to view a particular file, the displays for the other files can be switched off the screen and restored when they are needed. Any number of files can be edited during a given editing session. New library members can also be created without having to leave edit mode. For more information, see *Chapter 4. Editor.*

Procedures and Macros

Procedures (sometimes referred to as 'Command Lists') and macros are library members containing a sequence of VSE/ICCF statements that together carry out frequently-used functions. These functions could be, among others, compilation, loading and execution of programs, saving object decks and sorting library members. The statements in a procedure or macro can be system commands, editor commands, job entry statements, procedure or macro orders or data.

The purpose of such procedures and macros is to save you from having to re-enter the same series of commands and statements each time you want a particular function. Once the procedures or macros have been created and saved, you can later invoke them at any time simply by issuing a single command.

The main difference between procedures and macros is that a procedure can contain logical statements which determine the flow and execution of the commands, while a macro has only a limited logic capability.

Another difference is that macros are run in the foreground like normal commands and can be invoked while in edit mode, whereas procedures are run in the background and can only be invoked in command mode. Procedures require the procedure processor program to be executed in an interactive partition, which means that procedures are slower than macros. Variable parameter substitution is possible for both procedures and macros, which allows you to supply them with variable data, thus increasing their flexibility. The VSE/ICCF procedure processor includes the following capabilities:

- Branching in Procedures - Procedures can be constructed so that statements are skipped or repeated. The &&GOTO order causes the flow of lines in the procedure to be altered forward or backward.

- Conditional Statements - A conditional statement (&&IF order) can be included to cause a condition to be evaluated. Depending on the result of the evaluation, the flow of the procedure can be altered.

- Internal variable symbols - In addition to the nine parameters which can be entered from the procedure invocation (&&PARAM1 through &&PARAM9), nineteen internal alphameric variables (&&VARBLn) and nineteen internal numeric variables (&&COUNTn) can be used to store data or remember conditions. In addition, numeric variable symbols can be incremented.

- Prompting in a procedure - The &&TYPE order allows a procedure to write to a terminal, and the &&READ order allows a procedure to conversationally obtain data from the terminal.

- Reading data into procedures - The &&READ order can be used to read an entire line to be passed to the command processor as if it had come from the procedure itself. The &&READ order can also be used to assign a value to a variable (&&VARBLn) or a count field (&&COUNTn). In addition, the &&READ can also be used to read in new parameters to replace the original parameter (&&PARAMn) values.

- Return code interrogation - A procedure can also interrogate the outcome of a VSE/ICCF command or interactive partition execution.

- Punching data from a procedure - The &&PUNCH order allows a procedure to place data into the punch area.

A single macro can be created that contains groups of system or editor commands and run with one terminal operation, thus greatly extending editor flexibility. For example, a macro can be set up to place your terminal in edit mode, set tabs, indexing, verification and other defaults and return to the terminal to continue the edit process. Temporary macros can also be created during editing and run during the same session. The commands to be edited are placed in an area called the 'stack' area and run from there. Commands in the stack can be re-run any number of times.

IBM-supplied macros and procedures are made available with VSE/ICCF, but facilities are also provided in the system that enable you to write your own procedures and macros. A typical IBM procedure is ASSEMBLE, which assembles an assembly language program and produces an object module. Examples of IBM macros are: HC, which can be used to route the output to a hardcopy printer, or CPYLIB, which can be used to copy a member from one library to another. For a complete description of these facilities see *Chapter 3. System Commands, Procedures and Macros.*

Utility Programs

Besides the procedures mentioned in the previous section, which also carry out utility functions, VSE/ICCF provides specialized utility programs which perform certain frequently used, standardized functions like sorting, punching, and loading. Examples of such programs are:

LINKNGO - The LINKNGO utility is similar to the VSE linkage editor. It converts the output of language compilers and links programs and subprograms together to form an executable program.

OBJECT - The OBJECT utility transfers object programs and LINKNGO controls statements from the job stream to the punch area from where they are loaded into storage for execution.

DTSBATCH - You can use the DTSBATCH utility to enter VSE/ICCF commands off-line via SYSIPT or SYSLOG. The commands are processed against the VSE/ICCF library file as if they had been entered at the terminal. For example it allows you to update a library member while VSE/ICCF is down.

DTSCOPY - The DTSCOPY utility copies one sequential file to another.

DTSDUMMY - This program performs no processing. It is used in procedures where you might sometimes want to run a program and sometimes not, depending on variables.

DTSSORT - The DTSSORT utility performs a fast 'in storage' sort on data from the input or punch area.

DTSFDUMP - With the DTSFDUMP utility program you are able to interpret and format VSE/ICCF tables and storage locations for problem determination.

DTSAUDIT - The DTSAUDIT utility program allows you to trace changes made to the VSE/ICCF library file on the basis of a member, a group of members, an entire library, a group of libraries or all libraries.

Except for DTSFDUMP (which is described in *VSE/ICCF Installation and Operations Reference*), the above utilities are described in *Chapter 8. Utility Programs.*

Other VSE utilities are available to you that can be either run in an interactive partition or submitted to a VSE partition.

Debugging Facilities

In addition to the debugging facilities contained in the language compilers themselves, you are also able to investigate program failures using the VSE/ICCF Dump Program.

This program is automatically invoked by the /OPTION DUMP job entry statement whenever a program terminates abnormally. It displays various items of information about the termination, such as PSW, the general registers, data fields associated with the terminating instruction, etc. You can then enter other commands to display various storage areas within your program to locate the source of the problem. If desired, a hardcopy dump of an interactive partition can be obtained for later problem determination. For more information see *Chapter 6. Dump Commands.*

Print-Type Members

Print output from batch or interactive executions can be placed into the VSE/ICCF library. If the members containing such data are saved with names ending in '.P', they are considered print-type members (no other members should therefore end in '.P'). These members can be displayed or printed in print format with the /DISPLAY and /LIST commands, the PRINT macro and the RELIST macro. The print area ($$PRINT) is also considered to be a print-type member.

Members of this type contain control characters in the first two bytes of each record. VSE/ICCF recognizes these characters and displays or prints the output in print format. If the member name does not end in '.P', VSE/ICCF ignores these characters and each record is displayed on a new line, which means a line size of 80 characters. When a print-type member is printed on a hard-copy printer, or displayed on the IBM 3278 Model 5, up to 132 characters per line can be printed or displayed.

DBCS Support

If an IBM 5550 with 3270 Emulation is installed at your location, you can edit and display characters that have a one-byte representation like alphanumeric data *and* characters that have a two-byte representation like Japanese Kanji characters. If a character string contains characters of both a one-byte and a two-byte representation, we speak about *mixed data.*

Mixed data are structured into subfields of 'one-byte characters' and 'two-byte characters'. The beginning of a subfield of two-byte characters is indicated by a control character, which we call *shift-out (SO) character*, and the end by a control character, which we call *shift-in (SI) character*. When mixed data is displayed on the screen, the SO and SI characters occupy one position and the double-byte characters two positions. Depending on the terminal setup, the SO and SI characters are displayed as a blank or as any other special character. Internally, they are represented by X'0E' and by X'0F'.

VSE/ICCF allows you to process mixed data in various ways. You can

- Display on the screen and print on a hardcopy printer
  - VSE/ICCF library members
  - VSE/POWER list queue data
  - Print-type data produced by programs running in interactive partitions
- Edit mixed data with the full screen editor
- Do full screen read/writes from interactive partitions using the full screen macro DTSWRTRD
- Prepare mixed data for printing on an IBM 3200 printer.

If you want to create a library member that contains mixed data, you have to specify a special attribute, the *DBCS attribute*, for this member. The system command /PROTECT and the editor command SET allow you to set the DBCS attribute. Mixed data is recognized and interpreted as mixed data under the following conditions:

- *VSE/ICCF library members* are treated as mixed data if the DBCS attribute has been set for the member and if the terminal is an IBM 5550 with 3270 Emulation.
- *The print area* is treated as a DBCS member if it is displayed on an IBM 5550 with 3270 Emulation.
- Data in *the input area* is generally treated as alphanumeric data. You can, however, edit mixed data in the input area. To do this, you have to
  - Work at an IBM 5550 with 3270 Emulation
  - Edit the input area with the full screen editor
  - Temporarily SET the DBCS attribute for the input area.

  When you have finished editing and you save the input area as a member, the DBCS attribute becomes permanent for the saved member.
- VSE/POWER list queue entries and data for hardcopy output are treated as mixed data, if they are displayed/printed on an IBM 5550 with 3270 Emulation.

Mixed data may be misinterpreted and displayed as alphanumeric data, because

- An SO or SI character is not matched by its counterpart.
- The number of bytes between a pair of SO and SI characters is not even.

If mixed data is not displayed on an IBM 5550 with 3270 Emulation, double-byte characters will be displayed by meaningless symbols.

Display and Hardcopy Facilities

If you are using a display terminal you have various ways of displaying and printing the output from jobs in interactive partitions. One way is to write the total print output from a job, or job step, to a library member while it is being displayed.

Such a member can be a print-type member, which would also allow you to display, or print it in print format rather than in 80-character record format.

Another way of obtaining print output is to first save it in your library, after which you can direct it to the printer via the RELIST procedure. The data is queued on disk and printed automatically, thus freeing your terminal for other work during printing. You can also direct the output to a private queue or destination and then have it printed later.

You can also direct this print data to a hardcopy printer associated with the terminal via the @PRINT macro. The printer destination must be indicated by a /HARDCPY command issued during the session. For 328x terminal printers with the form feed feature, a simple form feed support (skip to next page) helps you format the print output that you receive from VSE/POWER. A CICS/VS service is used to print data on a hardcopy printer and VSE/POWER is used to route data to the system printer.

To simply display print data, the /LIST and /DISPLAY commands can be used. All of these commands, procedures and macros allow you to display, or print, data in print format -- provided the member has been created as a print-type member.

Chapter 2. Terminal Considerations

This section tells you something about setting up your terminal for operation, and how to log on to, and log off from VSE/ICCF. It also tells you about controlling output at the terminal, and special considerations which apply to particular terminal types.

There are basically two types of terminals that can be used with VSE/ICCF: typewriter terminals and display terminals. Within these types of terminals there are many different models including the following:

- IBM 3270 Display System via local attachment.
- IBM 3270 Display System via remote attachment.
- The 3284/3286/3287/3288/3289 printers can be attached to the above configurations and are supported for buffered hardcopy output.
- The IBM 2740 Selectric Typewriter Terminal is supported via leased or switched line remote attachment.
- The IBM 2741 Selectric Typewriter Terminal is supported via leased or switched line remote attachment.
- The IBM 3767 Terminal is supported in 2740 or 2741 mode.

The IBM 2740 (or 3767 in 2740 Mode)

Terminal Setup

Before attempting to log on to the system, ensure that the following steps have been taken:

1. Turn the terminal power on.
2. Set the LOCAL vs. COM switch to COM.
3. If the terminal is connected to a telephone rather than directly to a line, establish the dial connection as instructed by your VSE/ICCF administrator. When you hear the data tone, press the data button on the modem or place the hand set in the acoustic coupler and turn it on.
4. If the terminal is not connected to a telephone, ready the modem.
5. The Stand-by light should now be on.

6. On unbuffered terminals (model 1), press the BID key and wait for the Transmit light to turn on. Then press the EOB key. (If the auto EOB feature has been installed, the Carriage Return key can be used in place of the EOB key). On buffered terminals (model 2), press the BID key.

7. You can now log on.

Entering Data

Remember that the Backspace key can be used to correct data which has been typed incorrectly. When backspacing over any data characters, all characters which have been backspaced over must be retyped.

The Tab key can be used to request logical tabbing as specified in the /TABSET command. If a logical tab character has not been set via the /SET command, VSE/ICCF will consider the character transmitted when the Tab key is pressed as the logical tab character.

If the auto EOB feature is installed on the terminal, you need only press the Carriage Return key to enter a line of data. If the feature is not installed, press the Carriage Return key and the EOB key to signal the end of a line of input on an unbuffered terminal. To signal end of input on a buffered terminal, press the BID key.

On an unbuffered (model 1) terminal, the following steps must be taken to enter a command or a line of data. First, press the BID key. Second, wait for the Transmit light to turn on. Third, type your command or data line. Fourth, press the EOB key (or Carriage Return if auto EOB is installed) to terminate the transmission. The Transmit light should turn off and the Standby light should turn on.

On a buffered terminal (model 2), the following steps must be taken in order to enter a command or a line of data. First, press the ENTER key. Second, wait for the enter light to turn on. Third, type your command or data line. Fourth, press the BID key to signal that the buffer should be transmitted.

The IBM 3270

Terminal Setup

Before attempting to log on to the system, ensure that the following steps have been taken:

1. Turn the terminal power on. If you are using a printer also turn that on. The presence of the cursor on the screen will indicate that the power is on.

2. For local terminals, no other preparation is required.

3. For remote 3277 or 3275 terminals attached to leased lines (no telephone), make certain that the 'modem' is on and is ready.

4. If the terminal is connected to a telephone, establish the dial connection as instructed to do so by your VSE/ICCF administrator. (Dial the computer. When the data tone is heard, press the data button on the modem, etc.)

5. Press the CLEAR key. Then proceed with the logon procedure.

Entering Data

Data is always entered at the cursor position which will normally appear on the first line of the screen. To correct an error, simply position the cursor to the incorrect data and retype.

The default logical tab character on the 3270 keyboard is represented by the Field Mark key, which on the data entry keyboard is easy to access. However, on the normal typewriter keyboard it is in rather a difficult location for use with VSE/ICCF. It is upper shift PA2 key. The lower shift PA2 key is used as a Cancel key (see below) which makes it rather dangerous to use the key also for a Tab key. Therefore, on the 3270 terminal equipped with the typewriter keyboard, set the logical tab character (via the /SET command) to some other character; the semi-colon or the 'not' character are good choices.

To transmit data or commands via the 3270, you type in your data and then press the ENTER key.

If the INPUT INHIBITED light has been turned on (keyboard locked) due to some incorrect action by the user, it can be cleared by pressing the RESET key.

The ERASE INPUT key can be used to clear any data the user has just typed in. The CLEAR key, however, will cause the entire screen to be erased.

After the data has been entered and ENTER pressed, the data is sent to the computer and processed by VSE/ICCF. Depending on the nature of the input, one or more lines will be written back to the screen. Between the time the ENTER key is pressed and the response appears on the screen, the INPUT INHIBITED light will be on. No data or commands can be entered at this time.

Special 3270 Functions

The Program Access keys (PA1, PA2 and PA3) and the Program Function keys (PF1 through PF12, or sometimes through PF24) are used for special purposes in VSE/ICCF. The association of functions to the PA keys is done in the VSE/ICCF tailoring options. The default association is described here.

The Display key (PA3) displays the last ten lines before your current line in the input area, or in a library member. For example, when in edit mode, the Display key will display the ten lines up to and including the current line. In any other non-execution mode, the Display key displays the last ten lines of the input area.

In execution mode, the Display key is used as an Attention key if the program in execution has set an operator communication exit via the VSE STXIT OC macro. If no operator communication is in effect, the Display key will force any output in the print area. When the Display key is pressed during the display of the output, the remainder of the data in the print area will be lost.

The Cancel key is associated to the PA2 key. It can always be used to terminate the current mode of your terminal. The PA2 key serves no function when in system command mode. However ...

- When in input mode, the PA2 key performs the functions of the /END command and returns you to command mode.

- When in edit mode, the PA2 key performs the functions of the QUIT command and returns you to command mode. When in input submode of the editor, the PA2 key returns the terminal to the edit submode.

- When in execution mode, the PA2 key functions like the /CANCEL command. When a /DISPLAY or /LIST is in progress, the PA2 key will terminate the printout. When running in an interactive partition, the first PA2 request will request the cancellation of the job running in the background. Even though the job cancels, the remaining print output can still be displayed. However, pressing the PA2 key a second time while in this mode will cancel the remainder of the print output. For certain IBM-supplied subsystems (for example ISPF), pressing of the CANCEL key may not have the described effect. Instead the exit routine of the subsystem gets control.

Note that your installation may have assigned these functions to different keys. Perhaps they are assigned to PA1 and PA3, or perhaps the same keys were assigned but in the reverse order. Check with your VSE/ICCF administrator.

Logging On

The first step in using VSE/ICCF is logging on. The logon procedure starts the terminal session and identifies you to VSE/ICCF.

Note: *If you are using a dial-up terminal facility, first establish the connection with the installation according to rules given to you by your VSE/ICCF administrator. This usually consists of dialing a special phone number, waiting for the high pitched signal indicating that the computer has answered and then either placing your modem in data mode or placing the telephone handset into the cradle of the acoustic coupler and switching it on. Then in time you may be required to type in a 4-character terminal identifier to establish the connection between your terminal and the terminal control program (CICS/VS). In any case, these procedures should be obtained from your VSE/ICCF administrator.*

You can log on as soon as VSE/ICCF has been initiated at your installation. To check whether VSE/ICCF is in operation, press the ENTER key, Carriage Return key, EOB key, etc. A message like this should appear:

```
*INVALID TRANSACTION,  RE-ENTER REQUEST

       or

DFH2001 - INVALID TRANSACTION IDENTIFICATION - PLEASE RESUBMIT
```

If you are using a 3270, press the CLEAR key at this point before proceeding with the logon procedure.

To log on, type the following four characters and then press the ENTER or Carriage Return key:

```
iccf
```

The following message should be displayed:

```
*VSE/INTERACTIVE COMPUTING AND CONTROL FACILITY
*PLEASE ENTER /LOGON WITH YOUR USERID
```

Now type the following command, where xxxx is your user identification code, and press ENTER or the Carriage Return key:

```
/logon xxxx
```

If the user identification code was valid and known to the system, the following message will be displayed:

```
*ENTER PASSWORD
```

Now enter your password. If the password is valid and is accepted, any messages that have been sent to you will be displayed. The logon is completed by the following messages:

```
*LOGON COMPLETE - DATE 09/02/85 TIME 01:30
*READY
```

Your terminal is now in command mode and you can proceed with your terminal session.

Logon Procedure

Your user profile may indicate a macro or procedure that can be executed at logon time. (To check your user profile, see the /USERS command and the 'profile' option). This procedure can be used to set defaults, to display mail, identifiers or special instructions, or to carry out functions related to your logon procedure. You find a skeleton example in "Invoking the Editor" on page 4-9.

Getting Acquainted With VSE/ICCF

If you have just logged on for the first time and you want to practice with a few commands, enter:

```
/library
```

This will give you a list of the member names in your library. Or you might try:

```
/lib common
```

to obtain the list of member names in the common library, if one exists. You might pick one of the names and enter

```
/list aaaa
```

where aaaa is the name you selected. If your display does not end with the *READY message at this point, press ENTER (or EOB), or enter the /CONTINU command.

Some other commands which might be helpful at this point are:

```
/show
/mail
/time
/user profile
/user library
/user statistics
```

Problems During Logon

Here are typical problems that could occur during logon:

1. VSE/ICCF, or the system itself, is not running. This condition is generally indicated by no response to any ENTER or carriage return request from the terminal. You should check with someone at the installation.

2. Another user has left the terminal without logging off. You would thus receive the message *INVALID COMMAND, or some other VSE/ICCF response, when you attempt to enter your

/LOGON command. Enter the /LOGOFF command to log the previous user off, then repeat the logon procedure.

3. You have entered an invalid userid, which will be indicated by the *INVALID USER ID message. This means that the userid entered was not known to the system. There is no recourse in this situation; you must have a valid userid to log on. Enter /LOGOFF, /CANCEL, or press the PA2 key to terminate VSE/ICCF processing; then attempt to log on again with a valid userid.

4. If your userid has been accepted, you must also enter a valid password to complete the logon procedure. If you have forgotten your password, contact the VSE/ICCF administrator. Enter /LOGOFF to terminate VSE/ICCF processing.

Logging Off

It is very important that you log off after completing a terminal session; otherwise you are likely to be charged for more time than you have actually used.

To log off, simply enter the /LOGOFF command. Because this command is not valid in edit mode, you must first terminate the editor. If you want to save the contents of your input, punch or print areas, you must do this before issuing the /LOGOFF command; these areas will be cleared.

Terminal Input

All terminal input lines have a maximum length of 80 characters. However, multiple lines can be entered in a single terminal transmission. If you are using a 2741, 2740 or 3767, the Backspace key allows you to correct typing mistakes. You can also define a 'delete' character which, when keyed as the last character of a line, causes the entire line to be ignored (see the /SET command).

A Backspace key is not needed for a 3270, because you can position your cursor (with the left pointing Arrow key) back over the mistake and rekey only the characters in error. You can also enter a delete character, but it is more convenient to simply erase the entire input line before it is entered using the ERASE INPUT or CLEAR keys.

The /RETRIEV command allows you to retrieve previously entered commands. The retrieved command is placed in the terminal input area. You can reissue the command by simply pressing ENTER. Also, you can modify the command before you press ENTER.

Prompting

After you have entered the /INPUT command, your terminal is in input mode. When you are in this mode and you have set input verification off (see /SET VERIFY), you have the option of being prompted with the line number of the next line to be entered. The prompting number consists of four digits followed by a space. On typewriter terminals it prints to the left of the line being entered. On the 3270, the prompting number is displayed below the data to be entered.

If inclusion prompting is specified on the /INPUT command (or is the default), the prompt will be a five digit number. This is increased by a specified increment for each new line. Inclusion prompting causes the prompt to become part of the input data (in columns 1 to 5). The data which you type begins in column 7 of each line.

Normal or inclusion prompting are options which can be set on and off via the /PROMPT command. Or the prompt option may have been set up as a default in your user profile, or entered on the /INPUT command.

Tabbing

Logical tab positions are defined via the /TABSET (or edit TABSET) command. This allows you to press the Tab key (2740, 2741 and 3767) or enter a logical tab character for internal tabbing of the input data.

On typewriter terminals it is an advantage to have the physical tab stops on the unit set to the same positions as the logical tabs entered via the /TABSET command. You would use the /SET command to set the logical tab character.

The following example illustrates the use of tabbing and the logical tab character. Remember that, on the 2740, 2741 and 3767 terminals, the physical Tab keys can be used in place of a logical tab character:

```
*READY
/input
/tabset 7,73          (tabs are set for columns 7 and 73)
/set tab=;
;write (3,10)
10;format (' a = ?')
;read (1,20) a
20;format (f8.3)
;x=a**2
;write (3,25) a,x
25;format (' a = 'f8.3,' x = 'f20.3)
;end
/end
*READY
/list
       WRITE (3,10)
10     FORMAT (' A = ?')
       READ (1,20) A
20     FORMAT (F8.3)
       X=A**2
       WRITE (3,25) A,X
25     FORMAT ('A = 'F8.3,' X = 'F20.3)
       END
*END PRINT
*READY
```

If a backspace character is entered following a tab character, the effect of the tab is logically eliminated. The preceding example illustrates the use of the logical tab character (;) and the logical tab settings (column 7). Note that the TAB key could have been used in place of the logical tab character.

Multiple Line Input

The logical line end character can be set by means of the /SET system command or SET editor command. This allows you to enter several commands or data lines in a single terminal transmission, as in the following example:

```
*READY
/set end=:
*CONTROL SET
*READY
/input:aaa:bbb:ccc:/end:/save abcmem
*SAVED
*READY
```

In the preceding example, the colon (:) is set as the logical line end character. Then the /INPUT command followed by three data lines are entered. Finally, the input mode is terminated and the lines are saved in the library under the name ABCMEM.

Being able to enter multiple lines in a single transmission has two primary benefits. First, the system has less work to do. And second, you can work more effectively, because you do not need to wait for a terminal response to each line.

Following the line end character with a backspace causes the line end character to be eliminated. However, once data characters (not backspaces) have been entered following a line end character, it is not possible to backspace to a point before the line end character.

Entering Hexadecimal Data

You may sometimes need to enter data characters for which no key exists on your terminal keyboard. These characters must be entered as hexadecimal digits - two hex digits per character. Edit mode provides two commands (ALTER and OVERLAYX) that allow easy entry of hexadecimal data. In edit mode you may view all or parts of your file in hexadecimal format (VIEW command) and you may apply your changes in hexadecimal format.

It is also possible (in any mode of operation except full screen edit mode) to enter hexadecimal data by defining the HEX control character as in the following example:

```
*READY
/set hex=#
/input
/insert objmod
/delete
#02REP 000124 0C147F0
#02END
/end
*READY
```

In this example, the pound sign (#) is defined as the hex control character. When the hex character occurs in an input line, the two succeeding characters are assumed to be hexadecimal digits that will be used to form a single EBCDIC character. Thus in the preceding example, the #02 character string is interpreted as hexadecimal 02. The character placed in column 1 of the input line will thus be the EBCDIC character equivalent of hex 02. To use unprintable characters in the member name (for security reasons, for example), the characters following the hex character must be greater than X'40'.

Note: *Hex characters less than X'40' are translated to a non-display character before they are sent to the screen, otherwise they could be interpreted as screen control characters. If you use the 3270 Insert or Delete character keys to change a line containing such hexadecimal characters, so that these characters change their position within the line, the non-display character can no longer be retranslated correctly and the resulting line will be incorrect. View the hex data in hexadecimal format to avoid this problem.*

Escape Character

When you have several control characters defined (tab, backspace, line end, line delete and hexadecimal), you may sometimes want to enter one of these characters as an ordinary data character. Define a logical escape character as in this example:

```
/set esc="
```

You can then use the escape character prior to any of the defined control characters to cause the character to be entered as ordinary data. For example, if the colon (:) is defined as the logical line end character, a colon character could be entered as follows:

```
14":45":30 results in 14:45:30
```

ENTER Key in Execution Mode

If the user is in execution mode and the execution is in progress (that is, no conversational read outstanding and no queued print output waiting) and the ENTER key is pressed, a message is displayed as follows:

```
*BG IN PROGRESS, INPUT IGNORED, aaaa, bbbb, cccc
```

where 'aaaa' is the number of execution units used by the job, 'bbbb' is the number of lines printed thus far and 'cccc' is the number of cards punched thus far.

This message will be followed by the last ten lines (3270) or two lines (hardcopy terminals) placed in the print area.

Continuous Output Mode

When VSE/ICCF is writing output to the terminal, it stops after each screen of data has been transferred to allow you to read your output. To receive the next output, press ENTER or the Carriage Return key.

For the 3270 Display Terminal you must always press ENTER to receive the next output. To avoid this inconvenience, you can place your terminal in continuous output mode by entering the /CONTINU command after receiving the first output screen. The entire output to the terminal will thus proceed automatically with only slight pauses between screens of data.

Output Compression and Truncation

Terminal output from most compilers and utility programs usually contains 120 character positions per line, some output reports contain up to 132 print positions in each line. Some terminals such as the 2740 and the 3270 (but not the IBM 3278 Model 5 wide screen) cannot accommodate such a wide print line. Therefore, the remaining portion of a print line is displayed in either of two formats:

1. 'printline format'.

   The remaining portion is truncated. It extends, invisibly, to the right of the screen. Note that your display is automatically set to printline format as soon as you receive a display from

   - /DISPLAY a print-type member (or the print area $$PRINT)
   - /LIST a print-type member (or the print area $$PRINT)
   - /LISTP list output from a batch execution
   - being in SP=spool mode (or RD=conversational read mode)

   SYSLOG messages, too, have their righthand portion truncated when your display is set to printline format.

   /LIST or /DISPLAY of a compressed member and /LISTX **do not** cause the automatic setting of printline format.

2. 'wrap-around format'.

   The remaining portion is displayed on the next line. This format is also known as '80-character format'. Typically, you would see a print line as two lines in 80-character format if you display a member of print data that is not a print-type member (that is, the member name does not end with '.P').

Both formats tend to make the output confusing and difficult to read.

There are three ways of alleviating this problem. The first is print line compression. The /COMPRES command can be issued when a list or execution is in progress. When compression is in effect, all occurrences of multiple blanks within the print line are reduced to a single blank.

A second means of alleviating the problem is by truncation. If a job is run with the TRUNC option, only 78 characters of the print line are sent to the terminal. All other characters within the print line are ignored. The TRUNC option is very useful with language compiler output, because the compilers usually reduce very well into 78 characters. There is also another advantage: the option can be set up ahead of time (/OPTION TRUNC) on a job entry statement which then stays with the job on the VSE/ICCF library file. In addition, the TRUNC option causes jobs to run faster, because the truncated portion of the output is not written to the disk or to the terminal.

The third way of handling wide output is only possible if you are using a 3270 terminal. This method allows you to shift a page of output to the right and left, thus allowing all of the output to be viewed while still retaining clarity. You may use PF key functions in their default setting, as described in the following section. This is not recommended, however, because whenever you get a display in printline format, these default settings are suppressed; you would have to reset them by entering the /SHIFT OFF command. (The string <==MORE==> in the scale line indicates that your display is in printline format.) It is highly recommended that you use the /SHIFT command as described on page 3-126.

3270 Program Function Keys

The 12 (or 24 for some keyboards) Program Function (PF) keys initially have default settings (see below). You can overlay these by setting the keys to a data or command line which you need frequently. You can do this once for command mode, for edit mode, for execution and conversational read mode and for list and spool mode.

If you do not have PF keys (including non-3270 terminals) you can still take advantage of the ability to set program functions by entering the /PFnn command instead of pressing the Function key.

- The /SET PF command can be used to set a given Program Function key to a given command or data line. Then when the PF key is pressed (or when the /PFnn command is entered) the command or data line is processed just as if it had actually been typed in.

  It is also possible to set a PF key to a portion of a command or data line. Then when data is entered followed by pressing the PF key, the data entered is combined with the PF key setting to form the actual command or data line which will be processed (see the /SET PF command description).

  To set PF keys and still use the default PF key meanings described below, you can suspend the overlayed meanings with /SET ... OFF. To later reinstate your Program Function key settings, you need only specify /SET ... ON.

- The default settings of the PF keys in list and spool mode let you page forward and backward in your print output. They also allow you to shift pages left or right to view lines which exceed 80 characters.

  You can page forward/backward and page left/right also by using the /SKIP and /SHIFT commands, respectively. In fact, for print data, these commands are preferable over the default PF key settings. As explained in the preceding section, whenever you receive the display of print data, the default PF key settings are suppressed (and you would have to reset them by issuing

/SHIFT OFF). Also, the /SKIP and /SHIFT offer superior functions. For print data, therefore, try to ignore the default PF key settings and set your PF keys to /SKIP and /SHIFT command functions. The /SHIFT command is described on page 3-126, the /SKIP command on page 3-130.

A summary of PF key settings for list mode (/LIST and /LISTP) and spool mode appears below. These settings change according to the width of the screen being used. Program functions set with the /SET PFnn command override these defaults.

LIST MODE PRINT DISPLAY				
DISPLAY POSITIONS		PAGE		
WIDE SCRN	NORMAL SCRN	PREVIOUS	CURRENT	NEXT
1 - 132	1 - 80 ==>	PF1/PF13	PF2/PF14	PF3/PF15
31 - 162	31 - 110 ==>	PF4/PF16	PF5/PF17	PF6/PF18
31 - 162	61 - 140 ==>	PF7/PF19	PF8/PF20	PF9/PF21
31 - 162	91 - 162 ==>	PF10/PF22	PF11/PF23	PF12/PF24

Figure 2-1. Default PF Key Settings in List and Spool Mode

The PF1 key has another use when your terminal is not in execution mode: it can be used in place of the ENTER key. In this case the screen is not erased before the output is written back to the terminal, which can be useful if you want to keep certain information on the screen. After using the PF1 key in this way, press the ERASE INPUT key to clear the input line before typing in any data.

3270 Screen Format

The standard 3270 screen format under VSE/ICCF is shown below:

```
             Col                                         Col
              1                                          80
            ----------------------------------------------
LINE  1     I               (Input Area)
      2     I   ...+....1....+...(Scale Line)....7...XX...YY
      3     I
      4     I  (Output Area)
      .     I
      .     I
      .     I
      .     I
      .     I
      .     I
      .     I
     24     I
            I
```

The default input area is one line; however, you can expand this to 2, 3 or 4 lines so that several command or data lines (separated by end of line characters) can be entered in a single terminal transmission (see the /SET SCREEN command).

The scale line always separates the input area from the output area. The scale line facilitates the entry of fixed format data. The scale line (YY in the above diagram) indicates the current mode as follows:

CM -	command mode.
IN -	input mode.
ED -	edit mode.
FS -	full screen edit mode.
LS -	list-in-progress mode. Press ENTER to obtain the next screen.
EX -	execution is in progress in the interactive partition or the user is displaying execution mode print output.
RD -	a conversational read is outstanding from the program that is running in the interactive partition.
SP -	spooling continuous list display in progress from interactive partition execution. Press ENTER to obtain the next screen.
MS -	message mode. Press ENTER to obtain next screen of messages.

The XX characters in the scale line usually contain the scale line characters . + for column numbers. However, these positions will contain 'MR' when you are executing commands from a macro or from a multiple line sequence of input and there are more commands or lines to be processed in the macro or input sequence.

3270 Split Screen Use

The IBM 3270 terminal lets you define two or more output areas on your screen (see /SET SCREEN command). You can thus display data in one area and switch to the other and continue processing commands. The data in the first area is retained on the screen while you are working in the second. This is useful, for example, while viewing diagnostics from a program compilation. You can keep the diagnostics in one area of the screen and switch to the other area to correct the errors.

Multiple screens are particularly useful when Program Function keys are available. These keys can be equated to the various /SET SCREEN commands, thus allowing you to switch conveniently from one area of the screen to another. See the description of the /SET SCREEN command for an example of multiple screens used with PF keys.

3278 Model 5. The IBM 3278 Model 5 screen can contain up to 132 characters per line, and VSE/ICCF is able to use this wide screen when the terminal is in 'SP' mode during a job execution in an interactive partition. The same is also true when a /LIST or /DISPLAY command has been issued for a print-type member or for the print area. The wide screen is also used when a /LISTP command is issued to obtain a display of VSE/POWER list output during a full screen edit session, and when the scale line is extended to 132 characters.

Chapter 3. System Commands, Procedures and Macros

VSE/ICCF Command Language

The following four chapters describe in detail the command language that you will use in working with VSE/ICCF. The commands are presented in the following order:

- The system commands (and IBM-supplied procedures and macros) in this chapter.
- The full screen editor commands (and the IBM-supplied editor macros) in Chapter 4.
- The job entry statements in Chapter 5.
- The dump commands in Chapter 6.

System Commands begin with a slash (/) followed by a one to seven character alphabetic command name. They have an immediate effect, that is, their function is performed immediately after the command is accepted.

Macros and Procedures (such as PRINT, SDSERV or SORT) are documented in this manual as if they were commands. Actually, a macro or a procedure is a group of commands, and possibly job entry statements, that are executed as a complete function. They are referred to by their names.

(Full Screen) Editor Commands and macros can only be entered when the terminal is in full screen edit mode. You set your terminal in full screen edit mode by issuing the ED macro. (The /EDIT command places your terminal in context editor mode. The context editor is described in Appendix B.)

Job Entry Statements are entered as if they were data. They are placed in job streams to direct the running of jobs. For example, each execution job step must be preceded by a /LOAD statement. This statement specifies what compiler or other program is to process the following data. Job entry statements begin with a slash followed by a 1 to 7 character statement name.

Dump Commands allow you to display registers and various areas of programs that terminate abnormally. They are entered in response to prompting from the interactive partition dump program: K404D ENTER DUMP COMMAND. To use dump facilities, you must specify the DUMP option.

Command Syntax

Spacing

All statements, commands and procedures must begin in column 1 on their input line. One or more blank characters must follow the command verb if additional operands are coded on the line. If, however, the logical line end character (see "Entering Multiple Commands" on page 4-9) is the first character typed after an edit command, no intervening space is required.

Delimiters

Operands are usually separated by spaces. Commas or parentheses are, however, accepted as valid delimiters. For example, all of the following are equivalent:

```
    /EXEC  C$BSCEXC  CLIST  PROGRAM DATA
    /EXEC  C$BSCEXC  CLIST(PROGRAM,DATA)
    /EXEC  C$BSCEXC,,CLIST,(PROG),(DATA)
```

Delimiters cannot be used to indicate a missing operand. Multiple delimiters have the same meaning as a single delimiter. Most commands do not require an indication of a missing operand since the nature of the operands (e.g., alphabetic vs. numeric) usually indicates the presence or absence of other operands.

Abbreviations

Most system, editor and dump commands can be abbreviated but macro names, procedure names and the majority of job entry statements cannot. The full form of system, editor and dump commands is accepted, but the minimum (shown in capital letters) is sufficient. Occasionally, an alternate command form is given that cannot be abbreviated (such as /CP for /CTLP).

Syntax Rules

The descriptions of command formats on the next pages have the following syntax rules:

1. Brackets [ ] are not to be entered; they simply indicate syntax rules. Brackets around the slash [/] in front of some commands indicate that these commands are also available in edit mode. The slash must be entered if the command is used as a system command. A slash must not be used if the command is used in edit mode.
2. The capitalized letters of a command or an operand are the minimum to be specified; the lower case letters are optional.
3. Operands described entirely by lower case letters represent variable information to be supplied by you.
4. Underscored letters indicate a default option. If an underscored choice is selected, it need not be specified when the command is entered.
5. Operands in a stack or operands separated by '|' denote choices one of which must be selected. However, if the stack is enclosed by brackets, one **may** be selected or none.
6. Comments to commands or operands are enclosed in parentheses (); they are not part of the command or statement.

For some special rules on calling procedures and macros, refer to page 7-5.

Passwords

You can enter a 4-character password when saving a member in the library. The password must begin with a nonnumerical character and must not be PRIV or PUBL. Any later access to the member must include the password.

DBCS Support

Some system commands allow you to process or display mixed data. In Figure 3-1 on page 3-4 these commands, and also the procedures and macros which allow you to process mixed data, are marked with a 'Y' in the 'DBCS SUPPORT' column. The commands, macros, and procedures labelled with 'N' in this column do not support the handling of mixed data and the commands, macros, and procedures labelled with 'I' operate independently of the type of data.

Format of Examples

1. **Typewriter Terminal Example**: The lines entered by the terminal user are shown in lower case letters; system responses are shown in upper case letters.

```
*READY                    shows the system's readiness to accept commands
/switch 12                shows the command as entered by the terminal user
*SWITCHED                 shows the reaction ...
*READY                           ... of the system
```

2. **Screen Terminal Example**: This example is in two parts. The screen on the left shows the display before ENTER is pressed. The screen on the right shows the system's response.

```
/switch 12_
 ...+....1....+....2....+....3....+....4....+....5....+ ..CM
*READY
       _
        ...+....1....+....2....+....3....+....4....+....5....+.. ..CM
       *SWITCHED
       *READY
```

Summary of System Commands, Procedures and Macros

COMMAND	FUNCTION	DBCS SUPPORT
$	see /RUN command	I
ASSEMBLE	Causes a library member to be processed by the VSE assembler.	I
/ASYNch	Places the terminal into asynchronous execution mode.	I
/ATten	Signals request to communicate with running job.	I
/CANcel	Terminates input, execution, or list mode.	I
COBOL	Causes a library member to be processed by the DOS/VS COBOL compiler.	I
/COMpres	Displays output to your terminal in compressed form.	Y
/CONNect	Logically connects a secondary library to the primary library.	I
/Continu	Places the terminal into continuous output mode.	I
COPYFILE	Creates a copy of a library member under a specified name in your primary library.	I
COPYMEM	Creates a copy of a library member under a specified name in a specified library.	I
/COUnt	Returns the size of a VSE/ICCF library member.	I
/CP	See /CTLP command.	I
CPYLIB	Copies a library member from one library to another.	I
/CTL	See /SET command.	I
/CTLP	Used to display and alter the status of jobs in VSE/POWER queues (VSE/ICCF administrator only).	I

Figure 3-1 (Part 1 of 7). Summary of System Commands, Procedures and Macros

COMMAND	FUNCTION	DBCS SUPPORT
$DA	Causes the DTSDA program to run in an interactive partition, which displays the status of VSE partitions.	I
/DELete	Deletes the last line entered in input mode.	I
/DISPC	See /DISPLAY command.	Y
/Display	Displays, with line numbers, the contents of the input, punch, print or log area, or of a library member.	Y
/DQ	Displays the contents of the VSE/POWER queues.	I
/ECHO	Displays the data specified as operand.	Y
ED	Invokes the full screen editor directly.	Y
/EDit	Places the terminal into context editor mode.	I
EDPRT	Places the terminal in full screen edit mode for editing of the print area.	Y
EDPUN	Places the terminal in full screen edit mode for editing of the punch (stack) area.	N
/END	Ends an input session normally.	I
/ENDRun	Ends an input session and runs the job in the input area.	I
/EP	See /ERASEP command.	I
/ERASEP	Removes a file from a VSE/POWER queue.	I
/EXec	Runs a job or a procedure.	I
FORTRAN	Causes a library member to be processed by the VS FORTRAN compiler.	I
GETL	Retrieves VSE/POWER list queue output and places it in a library member or in the print area.	I
GETP	Retrieves VSE/POWER punch queue output and places it in a library member or in the print area.	I

Figure 3-1 (Part 2 of 7). Summary of System Commands, Procedures and Macros

COMMAND	FUNCTION	DBCS SUPPORT
GETR	Retrieves a job enqueued in the VSE/POWER reader queue and stores it as a member of the VSE/ICCF library file.	I
/GRoup	Creates a generation member group in your library.	Y
/HARdcpy	Places an IBM 3270 into hardcopy output mode.	Y
HC	Switches an IBM 3270 terminal to hardcopy mode, executes a specified command, and returns to normal display mode.	Y
HELP	Displays how—to—use information on VSE/ICCF commands.	Y
/INPut	Places the terminal into input mode.	N
/INSert	Copies all or part of a library member, or work area, into the input area.	I
/LIBC	See /LIBRARY command.	I
/LIBrary	Displays library directory information.	I
LIBRC	Catalogs a VSE/ICCF library member into a VSE library.	I
LIBRL	Displays a member from a VSE library.	I
LIBRP	Punches a member from a VSE library.	I
/List	Displays the contents of a work area, or a library member, without line numbers.	Y
/LISTC	See /LIST command.	Y
/LISTP	Displays print output from the VSE/POWER list queue.	Y
/LISTX	See /LIST command.	Y
LOAD	Causes an object program to be loaded into storage and run.	I

Figure 3-1 (Part 3 of 7). Summary of System Commands, Procedures and Macros

COMMAND	FUNCTION	DBCS SUPPORT
/LOCP	Locates a character string in a VSE/POWER list file.	N
/LOGOFF	Terminates the session.	I
/LOGON	Starts the session.	I
/LP	See /LISTP command.	Y
/MAil	Displays general and individual mail.	Y
/MSG	Displays messages.	N
MVLIB	Moves a library member from one library to another.	I
/PASswrd	Changes the logon password.	I
/PFnn	Simulates the effect of a PF key.	I
PLI	Causes a library member to be processed by the DOS/VS PL/I optimizing compiler.	I
PRINT	Routes the contents of a library member, or of the input area, to a hardcopy printer associated with a 3270 terminal for printing.	Y
/PROmpt	Sets line number or inclusion prompting on or off.	I
/PROTect	Adds or removes a password from a member or changes the PRIVATE/PUBLIC status of a member.	Y
/PURge	Purges a member from the library.	I
RELIST	Routes the contents of the print area to the printer.	Y
/RENAMe	Changes the name of a library member.	I
/RENUM	See /RESEQ command.	Y
/REPlace	Replaces a library member with the contents of the input area.	I

Figure 3-1 (Part 4 of 7). Summary of System Commands, Procedures and Macros

COMMAND	FUNCTION	DBCS SUPPORT
/RESeq	Changes or adds sequence numbers in or to a library member.	Y
/RETriev	Retrieves a previously entered command to the terminal.	I
/RETURN	Returns the terminal user to the interactive interface of VSE/System Package (VSE/SP).	I
/ROUTEP	Routes a VSE/POWER output file to a printer or punch.	I
/RP	See /ROUTEP command.	I
RPGIAUTO	Invokes AUTO REPORT and compiles with RPGII.	I
RPGII	Causes a library member to be processed by the DOS/VS RPG II compiler.	I
RPGIXLTR	Invokes the RPGII DL/I translator.	I
RSEF	Calls the RSEF program, the RPG II Source Entry Facility.	I
/RUN	Runs the job in the input area.	I
/SAve	Saves the contents of the input area as a library member.	I
SCRATCH	Removes DISP=KEEP files from the dynamic space area after they are no longer required.	I
SDSERV	Displays the directory (sorted) of a primary, connected, or of the common library.	I
/SENd	Sends a message to the system operator, or to another terminal user.	N
/SET	Sets VSE/ICCF system controls, 3270 screen features, control characters, VSE/ICCF features and editor autoinsert feature.	I
/SETIme	Sets the maximum number of execution units a job is to use.	I

Figure 3-1 (Part 5 of 7). Summary of System Commands, Procedures and Macros

COMMAND	FUNCTION	DBCS SUPPORT
/SHIft	Moves the display of printline data left or right.	Y
/SHow	Displays the settings of various VSE/ICCF control factors.	I
/SKip	Skips forward or backward in displayed output.	Y
SORT	Sorts a library member and places the output as directed.	I
/SP	See /STATUSP command.	I
┬SPACE	Causes the DTSSPACE program to be run in an interactive partition, which displays the dynamic disk space allocation areas.	I
/SQueeze	Converts a member to compressed format.	Y
/STatus	See /SHOW command.	I
/STATUSP	Displays the status of an individual VSE/POWER job.	I
STORE	Saves the contents of the punch area as a library member.	I
SUBMIT	Submits VSE/ICCF or VSE job streams to be run in a batch VSE partition.	I
/SUMry	Gives a summarized display of a library member.	Y
/SWitch	Switches from one library to another.	I
/SYNch	Terminates asynchronous execution mode and resets normal execution mode.	I
/TABset	Sets the logical tab positions.	Y
/Time	Displays date/time and execution units in the last background execution.	I

Figure 3-1 (Part 6 of 7). Summary of System Commands, Procedures and Macros

COMMAND	FUNCTION	DBCS SUPPORT
/USers	Displays the number of VSE/ICCF users, the number of users logged on, and user profile information.	I
VSBASIC	Is used to compile and punch or compile and run a VS BASIC program, or load and run an object deck.	I
VSBRESEQ	Is used to resequence a VS BASIC source program.	I

Figure 3-1 (Part 7 of 7). Summary of System Commands, Procedures and Macros

$ Command (See /RUN Command)

ASSEMBLE Procedure

The ASSEMBLE procedure causes a library member to be processed by the VSE assembler.

```
ASSEMBLE      name1 [OBJ name2|OBJ *] [options]
```

name1	is the name of a member in the library containing the assembler language source program to be assembled.
OBJ name2	is to be the library member name of the object module resulting from the assembly. If more than 8 characters are specified, only the 8 first are taken. This member need not initially exist.
options	are one or more /OPTION job entry statement options that alter the way the assembler is processed (e.g., NODECK, LIST, etc.).

The object module will be placed in your punch area if the OBJ operand is omitted or if OBJ * is specified. If the OBJ operand specifies a member name ('name2' parameter), the object module will be saved in the library under this name. If the name already exists, you are asked whether the object module is to overlay the existing member.

The 'options' operand can be specified as one or more options which influence the assembler, such as XREF, LIST, SYM, etc. If the NODECK option is specified, no object module will be produced even if the OBJ operand is present. The NORESET option should be specified if you are stacking multiple object decks in the punch area.

The LOAD procedure can be used to load and run the object module produced by the ASSEMBLE procedure.

Examples:

1.
   ```
   assemble proga
   load * data *
   ```

2.
   ```
   assemble progb obj objmem xref norld
   load objmem jes fildef data inpdata
   ```

/ASYNCH Command

The /ASYNCH command places your terminal into asynchronous execution mode while a job is running in an interactive partition. You can thus do other terminal work, such as entering or editing data while your job is running.

```
/ASYNch
```

This command is only effective in execution mode. It cannot be used when a conversational read is pending, nor may you use it while displaying print output.

Normally your terminal is blocked while you have a job running in an interactive partition. However, with the /ASYNCH command you can disconnect your terminal from this execution and carry on with other terminal work.

For example, you may have a long compilation running (with a /EXEC or /RUN command). With the /ASYNCH command you can return your terminal to command mode and carry on with other work, such as editing. However, do not attempt to edit the input area, or any member involved in the execution job stream.

When you have finished editing, issue the /SYNCH command. If the compilation is finished, the display will start. It is not possible to run a second job while your terminal is in execution mode.

While in asynchronous execution mode you can enter the /SHOW EXEC command. This will tell you if your job has reached a point where it requires the terminal.

Example:

```
*READY
/ex compile
*RUN REQUEST SCHEDULED FOR CLASS=A
/asyn
*ASYNCHRONOUS EXECUTION MODE ENTERED
ed memba
  .
  .   (edit commands)
  .
*READY
/syn
 (job output)
```

/ATTEN Command

The /ATTEN command signals that you want to communicate with a job that is running in an interactive partition.

```
/ATten       [data]
```

data the data to be passed to the program.

This command is effective only in certain situations. These are: in execution mode while a program is running, while a conversational read is outstanding, or while job output is being displayed.

The /ATTEN command can also be invoked on 3270 terminals by pressing the Display/Attention key[1]. On 2741 terminals the attention key has the same function.

If the program has an operator communication exit set (VSE STXIT OC macro), the /ATTEN command will interrupt execution and the exit is entered. If the /ATTEN command is entered while print output is being displayed, any remaining print data in the print area will be bypassed. If the /ATTEN command is entered while a conversational read is outstanding, any data supplied to the command as an operand is used to satisfy the conversational read. (This does not apply to the PA1 key, since no data can be passed with a 3270 PA key.)

If the /ATTEN command is entered while a program without an operator communication exit set is producing print data that has not yet been displayed, the accumulated print output is displayed. Remember, however, that this feature reduces the amount of data retained in the print area. Later print output will be written from the beginning, thus overlaying part or all of the print area's previous contents.

Example:

```
*READY
/run ditto
*RUN REQUEST SCHEDULED FOR CLASS=A
DITTO FUNCTION
?   (request conversational input)
fsr,280,1000
/atten  (interrupt execution)
DITTO FUNCTION
?
```

[1] The definition of the Display/Attention key is a VSE/ICCF tailoring option. The default is PA3. It may be different for your system.

/CANCEL Command

The /CANCEL command terminates whatever activity is in progress at your terminal. It is effective in input, list and execution mode.

```
/CANcel      [ABort]
```

ABort need only be specified for programs which use the /CANCEL command as an attention signal. The ABORT operand indicates that a cancel is requested instead of the attention signal.

Depending on the mode in effect when it is entered, /CANCEL produces the following effect:

Input Mode - The input session is terminated, the input area is cleared and command mode is entered.

List Mode - When a foreground job is in progress, /CANCEL terminates printing and returns you to the mode in effect before the display request. For example, if your terminal was in command mode when you entered a /LIST command, it would return to command mode.

Execution Mode - When a background job is in progress, or when print output is being displayed, the first /CANCEL command cancels it. However, the remaining data from the print area continues to be displayed. A second /CANCEL also terminates the display.

Notes:

1. *The /CANCEL command can also be entered as the response to a request for terminal input from a conversational program. In this situation, the background job which issued the request is canceled.*

2. *While a display is in progress, the terminal halts after each screen has been transferred. A /CANCEL command can be issued at this time. Even if the terminal is in continuous output mode (see /CONTINU command), there should be enough time between screens to enter the /CANCEL command.*

3. *If /CANCEL is issued while output from a procedure is being displayed, you may have to skip to the end (/SKIP END) and then issue /CANCEL. This would bypass the print output.*

Example:

```
*READY
/list longmem
FIRST OUTPUT LINE
SECOND OUTPUT LINE
(pause)
  .
  .
/cancel
*END PRINT
*READY
```

COBOL Procedure

The COBOL procedure causes a library member to be processed by the DOS/VS COBOL compiler.

```
COBOL        name1 [OBJ name2|OBJ *] [CBL] [options]
```

name1	is the name of a member in the library containing the COBOL language source program to be compiled.
OBJ name2	is to be the library member name of the object module resulting from the compilation. This member need not initially exist.
options	are one or more /OPTION job entry statement options that alter the way the compiler is processed (e.g., NODECK, LIST, etc.).
CBL	specifies that you are to be prompted for entry of a COBOL CBL option statement.

The object module will be placed in your punch area if the OBJ operand is omitted or if OBJ * is specified. If the OBJ operand specifies a member name ('name2' parameter), the object module will be saved in the library under this name. If the named member already exists, the object module will replace it.

The 'options' operand can be specified as one or more options which influence the compilation, such as XREF, LIST, SYM, etc. If the NODECK option is specified, no object module will be produced even if the OBJ operand is present. The NORESET option should be specified if you are stacking multiple object decks in the punch area.

The LOAD procedure can be used to load and run the object module produced by the COBOL procedure.

Examples:

1.
   ```
   cobol cobprg
   load *
   ```

2.
   ```
   cobol xxprog obj objmem list
   load objmem jes fildef data *
   ```

/COMPRES Command

The /COMPRES command compresses output before it is displayed or printed. It may only be entered in list or execution mode.

```
/COMpres      [ON|Off]
```

ON sets compression on. Specifying no operand (or any operand except 'OFF') has the same effect.

Off sets off space compression.

When operating upon alphanumeric data, space compression reduces all multiple space (blank) characters to a single blank character. When operating upon mixed data, space compression reduces all spaces that occur in subfields of double-byte characters to a double space. The /COMPRES command, however, has no effect if a DBCS member or the print area is displayed with the /LISTX command on an IBM 5550 with 3270 Emulation.

This command is useful for terminals whose line widths do not accommodate a full print line. Some printouts lend themselves well to space compression; others may yield better results with the TRUNC option of the /OPTION statement.

Even if some of the data has already been printed, you can still print a report or display data in compressed format. Just enter the /COMPRES command followed by a /SKIP command with a negative operand. This will back up the output to the beginning.

Example:

```
*READY
/list anymemb
A   B   C
D   E   F
/compres
G H I
J K L
```

/CONNECT Command

The /CONNECT command logically connects another library to your primary library. It is effective in command mode.

```
/CONNect        lib-no|OFF
```

lib-no	is the library identification number of the library to be connected to your primary library.
OFF	terminates the effects of a previous /CONNECT, thus leaving no library connected.

A connected library is scanned whenever you request access to a member that cannot be found in your primary library.

You can only connect to public libraries, and to those private libraries that are specified as alternate libraries in your user profile. The VSE/ICCF administrator can connect to any library.

You cannot update members in a connected library. Thus commands such as /EDIT, /RESEQ, etc. will fail if the member is in a connected library. To update these members, use the /SWITCH LIBS command to make the connected library your primary library and vice versa.

/LIBRARY or /LIBRARY CONN commands display the identification numbers of your primary and connected libraries (if the latter exist).

Notes:

1. *If the /CONNECT command is used within a procedure, it is only effective until the procedure has ended.*
2. *If a procedure includes a member from a connected library via a /INCLUDE, the /CONNECT command must be given prior to calling the procedure.*

Example:

```
*READY
/list cmemb (CMEMB is in library 5)
*FILE NOT IN LIB
/connect 5
*CONNECTED
*READY
/list cmemb
RECORD 1
RECORD 2
*END PRINT
*READY
```

/CONTINU Command

The /CONTINU command causes output to the terminal to be dislayed continuously. It is valid in list and execution modes, and ends when you leave these modes.

```
/Continu      [Off|nn|1|6]    (6 for 3270 only)
```

Off terminates the continuous display.

nn sets the pause between screens to 'nn' seconds, where 'nn' can be a value from 1 to 255. The default is 1 second, except for the IBM 3270, which is 6 seconds.

Normally, the terminal halts while each screen of data is being displayed. To receive the next screen, press ENTER or the carriage return key. During one of these pauses you can also enter the /CONTINU command, or some other command to direct the progress of the display.

Continuous output display with the time delay factor is very useful with the IBM 3270 Display terminal. You can adjust the rate of paging to your own convenience.

During continuous output, a short pause occurs between screens of output. These pauses allow you to issue the /CONTINU OFF or /CANCEL commands. Continuous output is also stopped if the editor is entered, or when the /LIST or /LIB commands are entered in asynchronous execution mode.

Example:

```
*READY
/exec printjob
*RUN REQUEST SCHEDULED
  . (output from job)
/con    (entered at first break, or before the first data is displayed)
  . (continuous output proceeds)
*****JOB TERMINATED - ICCF RC 00, EXEC RC 0000, NORMAL EOJ
*READY
```

COPYFILE Macro

The COPYFILE macro creates a copy of a VSE/ICCF library member which is saved in your primary library under another name. It is valid only in command mode.

```
COPYFILE     name1 name2 [pass]
```

name1	is the name of the member which is to be copied.
name2	is the name which will be given to the copy of the original member.
pass	is a four character password which applies to name1 and/or which will be applied to name2.

If copying cannot be completed because 'name2' is already in your primary library, you do not need to run COPYFILE again. The new copy is already in your input area. Either enter the /REP command to replace the 'name2' member, or save the input area using a different name from 'name2'.

Example:

Make a copy of the member called STATFIL and call the copy STATFILB:

```
COPYFILE STATFIL STATFILB
```

COPYMEM Procedure

The COPYMEM procedure creates a copy of a VSE/ICCF library member and saves it under a specified name in a specified library.

```
COPYMEM     name1 [pass1] lib1  name2 [pass2] lib2 [PUrge]
```

name1	is the name of the member which is to be copied.
pass1	is a four character password which applies to member name1.
lib1	is the number of the library in which member name1 resides.
name2	is the name which will be given to the copy of the original member.
pass2	is a four character password which applies to member name2.
lib2	is the number of the library into which member name1 is to be copied.
PURGE	causes the member name1 to be purged from library lib1.

The /CONN OFF command must be issued before you want to copy a member from/to a connected library. The libraries specified must be public or private libraries to which you have access according to your user profile.

Example:

Copy member MEMA with password PAS1 from library 3 to library 7, call the new member MEMB; give it the password PAS2, and purge MEMA from library 3.

```
COPYMEM MEMA PAS1 3 MEMB PAS2 7 PU
```

/COUNT Command

The /COUNT command lets you know how many records a VSE/ICCF library member consists of. It is valid in command mode only.

```
/COUnt        [CONN|COM] name
```

CONN	The connected library is to be searched for the specified member.
COM	The common library is to be searched for the specified member.
name	Specifies the name of the member. The names of the temporary areas $$PRINT, $$PUNCH, $$LOG, and $$STACK must not be specified as 'name'.

If neither CONN nor COM is specified, your primary library will be searched for the specified member.

When the member is found, the following message is displayed

```
   nnnnnn RECORDS IN MEMBER xxxxxxxx   [text]

where 'text' is: ** COMPRESSED **
             or: **  UPDATE IN PROGRESS **
             or: blank
```

/CP Command (See /CTLP Command)

CPYLIB/MVLIB Procedures

The CPYLIB procedure copies a member from one library to another. The MVLIB procedure performs the same function except that the member is purged from the original library.

```
CPYLIB      name [pass] lib1 lib2
MVLIB
```

name	is the name of the library member to be copied or moved from one library to another. The attributes of the already existing member are not transferred to the new member.
pass	password to be specified if the member is password protected.
lib1	is the library number which contains the member.
lib2	is the library number into which the member is to be copied or moved.

The libraries specified must be public or private libraries to which you have access according to your user profile. If library lib2 already has a member with the specified name, the existing member will not be overwritten.

Note: The /CONN OFF command must be issued first if you want to copy/move a member from/to a connected library.

Example:

Move member MEMBRZ from library 3 to library 7:

```
mvlib membrz 3 7
```

[/]CTL Command (See [/]SET Command)

/CTLP Command

The /CTLP command passes VSE/POWER commands to VSE/POWER. It can only be used in command mode, and is reserved for use by the VSE/ICCF administrator.

```
/CTLP       command
/CP
```

command	is a VSE/POWER command which can be used with the VSE/POWER cross partition communication. For example:

```
PALTER          PHOLD
PBRDCST         PINQUIRE
PCANCEL         PRELEASE
PDELETE         PXMIT
PDISPLAY
```

Which commands are actually valid depends on the VSE/POWER version your installation is using.

For a description of the VSE/POWER commands that can be passed with /CP, please refer to the *VSE/POWER Installation and Operations Guide*, SH12-5329.

Notes:

1. *To avoid conflicts, certain names should not be used; refer to the section "Submit-to-Batch Capability" on page 9-29.*
2. *Except for the display of queue directories, the VSE/POWER response to this command is limited to a single line.*

Example:

Provided your installation is a node of a VSE/POWER controlled network, this transmits a command to the remote node NODEX that will delete all jobs submitted by userid UUUU. NODEX must run with VSE/POWER.

```
/cp pxmit nodex,pdelete rdr,cuser=uuuu
```

$DA Procedure

The $DA ('Display Active') procedure displays the status of VSE partitions.

```
$DA
```

$DA loads the DTSDA program into an interactive partition and starts execution. The DTSDA program interrogates VSE control blocks and job accounting tables to display the status of VSE partitions. Partitions are displayed in order of dispatching priority: low to high. The units displayed should be considered relative and cannot be used for performance measurements.

Press the ENTER key to refresh the display, and enter any non-blank character to end it.

Note: To make the best use of this function, job accounting should be active in your system (JA=YES in the IPL command SYS). Also, // JOB statements should be supplied for all jobs, including those bring-up jobs for VSE/ICCF and VSE/POWER which can be initiated from procedures via the VSE Automatic System Initialization (ASI). All jobs should be ended by a /&, which ensures proper job accounting and a more accurate display. However, the $DA procedure does not show if a partition has been deactivated by the page manager because of heavy paging activity.

Example:

```
ID PIB JOB NAME   PHASE NAME **CPU** **ELAPSED** **SIO** TIME:17:09:14

F4  80 PARTITION UNBATCHED       *          *          *
BG  00 PAUSE     NO NAME      3.42   00:03:52        544
F2  00 ICCF      DTSINIT       .07   00:01:44         35
F3  00 NO NAME   NO NAME        *          *          *
F1  00 POWER     IPWPOWER      .61   00:04:21        461
```

ID	the VSE partition identification, displayed in ascending dispatching order (low to high).

PIB	VSE program information block hexadecimal state flag,
	`X'00' = active`
	`X'80' = unbatched`
	`X'82' = stopped`
	(for a detailed explanation refer to the *VSE/Advanced Functions Diagnosis Reference: Supervisor,* LY33-9107).

JOB NAME	the VSE 8-character job name, or `NO NAME` if the VSE partition is inactive or the // JOB statement was not given; `PARTITION UNBATCHED` if the PIB flag is X'80'; `PARTITION STOPPED` if the PIB flag is X'82'.
PHASE NAME	the VSE 8-character phase name of the // EXEC statement, or NO NAME if no phase is active at the moment.
CPU	the elapsed processor time in 1/100th seconds used by the current job step. It does not include overhead or all-bound time. (An * will appear in this column if your VSE does not include job accounting.)
ELAPSED	total elapsed time of the job in hh:mm:ss. A // JOB statement must have been given, otherwise a value for the elapsed time will not appear. (An * will appear in this column also if the partition is not working.)
SIO	the number of SIOs for this job step.
TIME	the time in hours:minutes:seconds.

/DELETE Command

The /DELETE command deletes the input line most recently entered into the input area.

```
/DELete
```

The /DELETE command is only valid in input mode. Successive /DELETE commands will erase successive previous input lines.

Example:

```
*READY
/input
aaaaa
bbbbb
/delete
ccccc
ddddd
eeeee
/delete
/delete
/end
*READY
/list
AAAAA
CCCCC
*END PRINT
*READY
```

/DISPC Command

/DISPLAY Command

The /DISPLAY command allows you to display the contents of the input, punch, print and log areas, and of a non-compressed library member. The data to be displayed can be alphanumeric or mixed data. It is valid in command and input mode.

```
/Display      [m [n]] [name [password]]
/DISPC
```

/DISPC	displays the file in continuous mode without requiring the /CONTINU command.
m	is the first line number to be displayed. If omitted, '1' is assumed. The maximum is '99999'.
n	is the last line number to be displayed. If omitted, or if it exceeds the number of lines in the file, the display will proceed through end-of-file. The maximum is '99999'.
name	is the name of the library member to be displayed. If name is omitted, the input area is displayed. Name can also be $$PUNCH, $$PRINT or $$LOG to display these areas.
password	is the password for the member. It need not be specified if the member is not password protected.

Line numbers appear at the left of each line. To display a part of a file, specify the first and last line number of the part to be displayed. If only one number is specified, it is assumed to be the starting line number. The file will then be displayed from this line to the end of the file.

For a print-type member, the first two bytes of each record are interpreted as print control characters, and the member is displayed in printline format. The default PF key settings (as described in Figure 2-1 on page 2-12) are **not active**; you can use your own PF key settings, instead.

Notes:

1. *You can use the /SKIP command to skip over certain parts of the file.*
2. *The /DISPLAY and /LIST commands have identical formats and nearly identical functions, except that the /DISPLAY command also shows line numbers.*
3. *If columns 73-80 contain sequence numbers, the print lines are truncated at column 72, unless the display width (see /SET LINESIZE) has been altered from the default width of 72. If any other data appears in 73-80, it will be displayed regardless of the linesize setting. For print-type members the line numbers will be treated as print data.*
4. *When this command is used in input mode, your terminal will go into list mode until the display is complete. Then input mode will be re-instated.*
5. *Print-type members are displayed using all 132 characters of the IBM 3278 Model 5 wide screen.*
6. *On a hardcopy printer, a print-type member is printed using the total linesize.*

7. *For print-type members, line numbers refer to print records, which are not identical with physical records.*

8. *The records in a print-type member that do not contain valid print control characters in the first two columns are printed as 80-byte records.*

Example:

```
*READY
/input
first line
second line
third line
/display
0001 FIRST LINE
0002 SECOND LINE
0003 THIRD LINE
*END PRINT
fourth line
fifth line
/end
*READY
/save sampl1
*SAVED
*READY
/d sampl1
0001 FIRST LINE
0002 SECOND LINE
0003 THIRD LINE
0004 FOURTH LINE
0005 FIFTH LINE
*END PRINT
*READY
```

/DQ Command

The /DQ command displays the contents of the VSE/POWER list, punch, reader and transmit queues.

```
/DQ           [ALL|*abc|FREE|HOLD|LOCAL]
              |[queue[ALL|*abc|FREE|FULL=YES|HOLD|
                     LOCAL|jobclass|jobname [jobnumber]]]
```

queue	specifies the queue for which the /DQ command is intended: LST for list queue PUN for punch queue RDR for reader queue XMT for transmit queue If 'queue' is not specified, status information is displayed for all jobs in all queues.
ALL	gives the status of all jobs in the specified queue. If 'queue' is not specified, status information for all jobs in all queues is displayed. ALL is the default.
FREE	gives the status of all jobs in the specified queue that are available for processing. These are the jobs that are in the keep or dispatchable state.
FULL=YES	causes the display of all displayable information about a particular queue entry.
HOLD	gives the status of all jobs in the specified queue that are not available for processing. These are the jobs that are in hold or leave state.
jobclass	gives you the status of all jobs with this jobclass in the specified queue. Can be a character from A to Z, or from 0 to n, where n is the partition number (input jobclass only).
jobname	is the 2 to 8 character job name by which the job is known to VSE/POWER.
jobnumber	is the 1 to 5 digit job number assigned to the job by VSE/POWER.
LOCAL	gives the status of all jobs in the specified queue that were submitted from, or routed to your installation.
*abc	requests the status of all jobs whose names begin with the characters that you specify. For 'abc' you can specify up to eight alphameric characters, including "/", ".".

The operands of the /DQ command are identical to those of the VSE/POWER PDISPLAY command. Only a subset of all possible operands is shown here. Refer to the operand description of the PDISPLAY command in *VSE/POWER Installation and Operations Guide* for all valid operands.

Note: VSE/POWER diagnoses any operand errors in the /DQ command and displays appropriate messages.

Examples:

1. Display the contents of all VSE/POWER queues.

```
/dq_
 ...+....1....+....2....+....3....+....4....+....5....+ ..CM
*READY
```

```
_
 ...+....1....+....2....+....3....+....4....+....5....+....6. ..CM
1R46I READER QUEUE   P D C S  CARDS
1R46I CISTART  00696 3 * 2       68
1R46I JOB02    00527 3 D A      343 FROM=(AAAA)
1R46I   LIST QUEUE   P D C S  PAGES CC FORM
1R46I CISTART  00696 3 D A        3
1R46I JOB01    00439 3 L Q       12         TO=(AAAA) FROM=(AAAA)
1R46I  PUNCH QUEUE   P D C S  CARDS CC FORM
1R46I JOB01    00439 3 D A       12         TO=(AAAA) FROM=(AAAA)
*END PRINT
*READY
```

The directory information for the reader, the list and the punch queues is returned from VSE/POWER. For a precise description of this response see *VSE/POWER Installation and Operations Guide.*

The columns and their meaning:

1st column	:	'1R46I'	VSE/POWER message number.
2nd column	:	'xxxxxxxx'	VSE/POWER job name.
3rd column	:	'nnnnn'	VSE/POWER job number.
P - column	:	'p'	Priority.
D - column	:	'd'	Disposition.
C - column	:	'c'	Class.
S - column	:	's'	System id for shared spooling.
CARDS	:	'ccccc'	Number of input records or punch cards.
PAGES	:	'ppppp'	Number of pages in print output.
CC	:	'cc'	Copy count to be printed/punched.
FORM	:	'ffff'	Forms number.
FROM=(userid)			Submitter.
TO=(userid)			Addressee.

2. Display all displayable information of the VSE/POWER LST queue.

```
/dq LST FULL=YES_
...+....1....+....2....+....3....+....4....+....5....+ ..CM
*READY
```

```
_
...+....1....+....2....+....3....+....4....+....5....+....6. ..CM
1R46I   LIST QUEUE   P D C S  PAGES CC FORM
1R46I CISTART  00202 3 * A        3  1
        D=12/22/83, L=    62
1R46I CISTART  00204 3 D A        3  1
        D=01/11/84, L=    60
1R46I RELIST    00211 3 D A        1  1      TO=(AAAA) FROM=(AAAA)
        D=01/12/84, L=     6
1R46I RELIST    00213 3 D A        1  1      TO=(AAAA) FROM=(AAAA)
        D=01/12/84, L=     6
1R46I RELIST    00215 3 D A        5  1      TO=(AAAA) FROM=(AAAA)
        D=01/12/84, S=001, L=    245
1R46I RELIST    00215 3 D A        5  1      TO=(AAAA) FROM=(AAAA)
        D=01/12/84, S=002, L=    300
1R46I RELIST    00215 3 D A        5  1      TO=(AAAA) FROM=(AAAA)
        D=01/12/84, S=003, L=    300
*END PRINT
*READY
```

The columns have the same meaning as in Example 1. In addition, the following information is displayed:

D=	:		Date
L=	:	'lllll'	Number of lines
S=	:	sss'	Suffix number

[/]ECHO Command

The [/]ECHO command displays the operand data.

```
[/]ECHO        message
```

message is any text that you want written back to your terminal.

The ECHO command (without /) is effective in edit mode; the /ECHO command is effective in any other mode.

For 3270 terminals, the screen is not erased; the echo data replaces the first output line on the screen.

This command can be used in macros for displaying a completion message back to the terminal.

| Mixed data can only be displayed with the /ECHO command if the command is invoked from a macro
| and if the display station is an IBM 5550 with 3270 Emulation.

Example:

```
/echo operation complete
OPERATION COMPLETE
```

ED Macro

The ED macro directly invokes the full screen editor. It is valid in command mode.

```
ED          [name [password]]
```

name	is the name of the library member to be edited. If no name is specified, the input area is edited. The named member must be in your primary library.
password	can only be specified if name is specified and the member is password protected.

This macro is a /EDIT command followed by a SET DELAY BYPASS to display only the output from the last command of a multiple line input or a macro, followed by a VERIFY FULL to invoke the full screen editor. If you want to write your own full screen editor macro to set your PF keys, expand this macro accordingly.

/EDIT Command

The /EDIT command places the terminal in **context editor** mode. It is only valid in command mode.

```
/EDit          [name [password]]
```

name	is the name of the library member to be edited. The member must be in your primary library. If no name is specified, it is assumed that you want to edit in the input area.
password	is only required if the library member being edited is password protected.

If 'name' is specified, the edit commands apply directly to the member. Editor commands have an immediate effect on the data being edited. After you have finished editing in the input area you can issue the SAVE command to save your edited data as a member in the library. The SAVE command also returns you to command mode.

If the input area is empty when you enter the /EDIT command, you can only start typing in data after you have issued the INPUT command.

Notes:

1. *When editing from a local 3270, verify long mode is assumed. Thus, you see a full screen display of the part of the file you are editing. If you are using a remote 3270, issue VERIFY LONG to get a full screen display.*
2. *To enter full screen edit mode, the VERIFY FULL command must be issued after the /EDIT command.*
3. *Mixed data other than DBCS print-type members can only be edited in full screen edit mode.*

Example:

```
*READY
/input
first card
second card
third card
/end
*READY
/edit
*EDIT MODE
locate sec
SECOND CARD
ch /seco/2
2ND CARD
save sampl2
*SAVED
*READY
```

EDPRT/EDPUN Macros

The EDPRT and EDPUN macros are used to enter edit mode to view or edit the print and, respectively, punch (stack) area. They are valid only in command mode.

```
EDPRT
EDPUN
```

The EDPRT macro can be used to locate compilation errors within the print area. It places your terminal in edit mode, which allows you to use commands such as LOCATE to find particular data within the print area. The same approach can also be used to modify data before routing the contents of the print area to the system printer via the RELIST macro. Note, however, that additions or deletions of entire lines cannot be made using this macro.

The punch area can also be modified in this way before saving it in the library or before using it to form part of a job stream (via the /INCLUDE $$PUNCH statement).

/END Command

The /END command terminates an input session. It is only valid in input mode.

```
/END
```

The /END command closes the input area and places the terminal in command mode. To add further data to the input area you must enter edit mode.

If you are using an IBM 3270 you can also use the Cancel key[2]
to terminate the input session.

Example:

```
*READY
/input
card 1
card 2
card 3
/end
*READY
/list
CARD 1
CARD 2
CARD 3
*END PRINT
*READY
```

2 The definition of the Cancel key is a VSE/ICCF tailoring option. The default is PA2. It may be different for your system.

/ENDRUN Command

The /ENDRUN command has the same effect as an /END command followed by a /RUN command. It can only be used in input mode.

```
/ENDRun
```

This command terminates the input session and runs the job in the input area.

Example:

```
*READY
/input
/load vfortran
;write (3,10)   (note that ; is the logical tab character)
10;format (' this means the program ran')
;end
/endrun
*RUN REQUEST SCHEDULED
  (compiler output)
TOTAL MEMORY REQUIREMENTS 000154 BYTES
HIGHEST SEVERITY LEVEL OF ERRORS FOR THIS MODULE WAS 0
***** BEGIN LINKNGO
  12,236 BYTES REQUIRED FOR PROGRAM STORAGE
  PGM LOADED AT 09B100
  XFER ADDRESS 09B100
*****
THIS MEANS THE PROGRAM RAN
**** JOB TERMINATED - ICCF RC 00, EXEC RC 0000, NORMAL EOJ
*READY
```

/EP Command (See /ERASEP Command)

/ERASEP Command

The /ERASEP command removes a job from the specified VSE/POWER queue. It is valid in command mode only.

```
/ERASEP      queue jobname [jobnumber[ jobsuffix]][ PWD=password]
/EP          queue ALL
```

queue	specifies the VSE/POWER queue to which the command applies in the form:
	`RDR for reader queue`
	`LST for list queue`
	`PUN for punch queue`
	`XMT for transmission queue`

jobname	is the 2 to 8 character name of the VSE/POWER job that is to be erased.
jobnumber	specifies the 1 to 5 digit jobnumber assigned to the job by VSE/POWER.
jobsuffix	is the 1 to 3 digit job suffix which designates the segment number. If this parameter is omitted, the entire job will be erased.
PWD=password	indicates the 1 to 8 alphanumeric password given to the VSE/POWER job when it was submitted.
ALL	causes all jobs to be removed from the specified VSE/POWER queue. It can only be specified by the VSE/ICCF administrator.

If you request a job, identified by jobname and possibly jobnumber, to be removed and this jobname (and jobnumber) occurs more than once in the queue, the job occurring first will be removed.

When specified with 'RDR' and the jobname (and optionally the jobnumber and/or password), the /ERASEP removes a job from the VSE/POWER reader queue. You would use this command, for example, to cancel a job that you have decided not to run. If the job has already started execution, send a message to the system operator with the /SEND command asking him to cancel it for you.

When specified with 'LST' or 'PUN', /ERASEP removes the print or punch output of the named job from the specified VSE/POWER output queue. For example, if you have already viewed the output from a job, use this command to erase the output.

Note: This command has replaced the /PURGEP command. /PURGEP is still valid, but is not recommended.

Example:

Erase the punch output of 'MYJOB' from the VSE/POWER punch queue. The jobnumber is 0200 and the password is 'SECRET':

```
/erasep pun myjob 0200 pwd=secret
*OK
*READY
```

/EXEC Command

The /EXEC command runs a job stream, or a procedure, which is a member in your library. It is only effective in command mode.

```
/EXec       name [password] [CLIST [param1 param2 ...]]
```

name	is the name of a member in your library or of the punch area $$PUNCH. The member or the punch area must contain a job stream or a procedure. The job stream can consist only of job entry statements, or of job entry statements and data.
password	need only be specified if the library member is password protected.
CLIST	indicates that the member specified contains a procedure which must be processed using the DTSPROCS utility.
param1 ..	are parameters which are to be passed to the procedure being run. Parameters should only be specified if the procedure has been written using variable symbols in place of operands. The parameters specified on the /EXEC command will replace the variable symbols in the procedure.

A job stream to be run must begin with a /LOAD job entry statement. The job stream can consist of a single step or multiple steps. The job stream can consist only of job entry statements or it can consist of job entry statements and the data to be processed. Alternately, the data to be processed by the job can be in a different library member and can be referenced by the /INCLUDE statement. If the member is password protected the 'password' parameter must be specified.

Procedures are described more fully in *Chapter 7. Writing Procedures and Macros.* Basically, a procedure consisting of a simple set of /LOAD steps is no different from any other job stream. It can be run with a simple /EXEC command.

The /EXEC command is the equivalent to the following commands and job entry statements:

```
/INPUT
/INCLUDE name [password]
/END
/RUN
```

The command /EXEC $$PUNCH allows you to load and run an object deck (not BASIC) that is in the punch area. Normally the output of a language translator is placed in a punch area from which it is retrieved by the LINKNGO program for execution. Sometimes it may be useful to re-run the object module a second or third time without recompiling the source program. To do this, issue the /EXEC $$PUNCH command.

In the following situation, the CLIST form of the /EXEC command must be used. A procedure (command list) has been cataloged to the library that does not begin with the /LOAD DTSPROCS statement:

```
/EXEC PROC CLIST
```

If the procedure list has been defined with variable symbols in place of operands, the CLIST plus parameter form of the /EXEC command must be used. The parameters specified following the CLIST operand are substituted for the variable symbols when the procedure is run. For example, the command

    /EXEC BSCEXEC CLIST OBJMOD DTAMOD

might run the user-written procedure BSCEXEC. This procedure causes a BASIC object deck stored in the library (OBJMOD) to be loaded and run using the input data also stored in the library (DTAMOD). If the implied execute function has not been turned off, the /EXEC command name and the CLIST operand need not be specified. Thus, the preceding example can also be specified as follows:

    BSCEXEC OBJMOD DTAMOD

Notes:

1. *The /EXEC statement destroys the contents of the input area. To preserve the input area, you should enter a /SAVE or /REPLACE command prior to entering the /EXEC command.*

2. *After an /EXEC command has been entered, and execution has finished, the job can be rerun with a /RUN (or $) command. No operand is needed. An /INCLUDE for the member to be run will already have been placed in the input area by the processing of the /EXEC command.*

3. *When ENTER is pressed while execution is in progress, *BG IN PROGRESS is displayed together with the last print lines placed into the print area. The last two lines are displayed for typewriter terminals and the last ten for the IBM 3270.*

Examples:

1. Build a job stream in input area and execute it:

```
*READY
/input
/load basic
/option getvis=100
run *
010 print 'enter your first name'
020 input a$
030 print '   hello   ', a$
040 end
/data incon
/end
*READY
/save hello    (save the job stream)
*SAVED
*READY
/exec hello                 (run the job stream)
*RUN REQUEST SCHEDULED
(output of compile, load, and run)
```

2. Execute the procedure SDSERV:

```
*READY
/exec sdserv clist        (This procedure runs and displays a
*RUN REQUEST SCHEDULED    sorted list of your library)
   .
   .
(print output)
   .
   .
*END PRINT
```

3. Execute a procedure and pass parameters to it:

```
*READY
/exec myload clist myprog mydata  (This procedure loads the object
*RUN REQUEST SCHEDULED            deck MYPROG from the library and
   .                              runs it.  MYPROG reads its input
   .                              data from the member named MYDATA.)
(output)
```

FORTRAN Procedure

The FORTRAN procedure causes a library member to be processed by the VS FORTRAN compiler.

```
FORTRAN       name1 [OBJ name2|OBJ *] [PROCESS] [options]
```

name1	is the name of a member in the library containing the FORTRAN language source program to be compiled.
OBJ name2	is to be the library member name of the object module resulting from the compilation. This member need not initially exist.
PROCESS	specifies that you are to be prompted for entry of VS FORTRAN compiler options.
options	are one or more of the VSE/ICCF unique options of the job entry statement /OPTION, for example: ANYPHASE, CLEAR, CONTINUE (see page 5-24).

The object module will be placed in your punch area if the OBJ operand is omitted or if OBJ * is specified. If the OBJ operand specifies a member name ('name2' parameter), the object module will be saved in the library under this name. If the named member already exists, the object module will replace it.

The PROCESS operand must be specified if you want to set one or more compiler options which influence the compilation, such as XREF, LIST, SOURCE. If the NODECK option is specified, no object module will be produced even if the OBJ operand is present. The NORESET option should be specified if you are stacking multiple object decks in the punch area.

The LOAD procedure can be used to load and run the object module produced by this procedure.

Examples:

1.
   ```
   fortran fortest
   load *
   ```

2.
   ```
   fortran xxprog obj objmem
   load objmem jes filder data *
   ```

GETL Procedure

The GETL (GET LIST QUEUE) procedure retrieves output from the VSE/POWER list queue and places it in a library member, or in the print area.

```
GETL        jobname [jobnumber [jobsuffix]] [jobclass] [NOPRINT|PRINT]
                    [KEEP|DELETE] [MEM=member|*]
                    [PWD=jobpassword]
```

jobname	is the 2 to 8 character name of the spooled list output in the VSE/POWER LST queue which you want to retrieve.
jobnumber	is the 1 to 5 digit number assigned to the spooled list output in the VSE/POWER LST queue which you want to retrieve. In general, it is identical to the jobnumber of the job which produced the output (but see the Note below). It must be specified if 'jobname' is not unique within the LST queue. *Note:* The jobnumber in the list queue is different from the number of the job which produced the output in either of the following cases: 1. The job stream was submitted via an /INCLUDE with the ICCFSLI option **and** contained a * $$ LST statement. 2. The job had been run under control of the VSE/POWER PNET networking function.
jobsuffix	is the 1 to 3 digit job suffix which designates the segment number. If this parameter is omitted, segment number 1 is assumed.
jobclass	is the class (A-Z) of the spooled output in the VSE/POWER LST queue. It must be a class for which no printer has been started. The default output class is Q[3].
NOPRINT |PRINT	specifies whether print control characters are to be placed in the output data. PRINT causes the member to be created in the format required for processing by RELIST (see RELIST in this chapter). The default is NOPRINT.
KEEP |DELETE	specifies the disposition of the output in the VSE/POWER LST queue after retrieval is complete. KEEP is the default. If DELETE is specified, the job will automatically be deleted from the VSE/POWER LST queue (if the queue entry is segmented and you do not specify a jobsuffix, all segments of the queue entry will be deleted). Abnormal termination results in the output being kept.
MEM = member|*	specifies the name of the VSE/ICCF library member into which the list output retrieved from the VSE/POWER LST queue is to be placed. If no member (or 'MEM = *') is specified, the list output is placed into the print area. By specifying '/HARDCPY devicename queuename' before invoking the GETL procedure, VSE/POWER LST queue contents can be transferred to a hardcopy printer. Under CICS/VS, they can be transferred to a private print destination queue for hardcopy output.

[3] For your installation a different default may have been defined.

PWD=jobpassword	is the 1 to 8 alphanumeric character password that you assigned to the job and its output at submission time (see SUBMIT procedure) or, if you are allowed to submit your own VSE/POWER JECL, the password assigned to the list output according to the rules of VSE/POWER. You can omit it if there is no password protection, or if you are the VSE/ICCF administrator. *Note:* Jobs submitted via the card reader are protected by an unprintable password if you do not provide one. Output from such jobs can only be retrieved by the VSE/ICCF administrator.

To avoid transferring unnecessarily large files, always specify the point where you want retrieval to begin, and the number of lines that you want transferred. Message K871D, which appears after the GETL procedure has been invoked, allows you to do this. It tells you the size of the file that is to be transferred and asks how much you want to have transferred. For example, if you simply want to look at the errors in a large program before deciding whether to release the output to a printer, you will only need to view the last 50 to 100 lines.

The jobname is the only required operand. If the program DTSGETQ (the program that does the actual retrieving of the list output) is canceled for some reason, it is your responsibility to purge the library member that was created via MEM=member.

Notes:

1. *Only the submitter or the user designated in the DEST parameter of the VSE/POWER JECL statement * $$ LST may access the list output queue entry. An authorized user may specify ANY in the DEST parameter (authorization is established in the VSE/ICCF user profile). This makes the list output accessible to any user.*

2. *The GETL procedure invokes the DTSGETQ program. The DTSGETQ program runs in an interactive partition and processes the procedure based on the positional parameters 'jobname', 'jobnumber', 'jobsuffix', and 'jobclass'. These are the only positional parameters in the procedure and they must be specified as shown. They are separated by blanks. The other (keyword) parameters, including their given values, can appear anywhere in the procedure.*

3. *'CANCEL' cannot be used as a job name or password.*

4. *The list output to be retrieved must be in an output class for which no VSE/POWER task has been started.*

Examples

1. Place VSE/POWER list job 'MYJOB', jobnumber 0200 with password 'SECRET' in VSE/ICCF library member 'MEMBERA', and delete 'MYJOB' from VSE/POWER queue after successful transfer:

   ```
   getl myjob 0200 a pwd=secret mem=membera delete
   ```

2. Place VSE/POWER list job 'MYJOB' with password 'SECRET' in the print area. If the /HARDCPY command was correctly issued beforehand, VSE/POWER output is transferred directly to a hardcopy printer:

   ```
   getl myjob mem=* pwd=secret
   ```

3. Place VSE/POWER list job 'MYJOB' with password 'SECRET' in VSE/ICCF library member 'LISTJOB' with print control characters for processing by RELIST. Then delete the job from the VSE/POWER list queue:

   ```
   getl myjob mem=listjob pwd=secret print delete
   ```

GETP Procedure

The GETP (GET PUNCH QUEUE) procedure retrieves output from the VSE/POWER punch queue and places it in a library member, in the punch area, or in the print area.

```
GETP        jobname [jobnumber [jobsuffix]] [jobclass] [KEEP|DELETE]
            [MEM=member|*|$$PRINT] [PWD=jobpassword]
```

jobname	is the 2 to 8 character name of the spooled punch output in the VSE/POWER PUN queue which you want to retrieve.
jobnumber	is the 1 to 5 digit number assigned to the spooled punch output in the VSE/POWER PUN queue which you want to retrieve. In general, it is identical to the jobnumber of the job which produced the output (but see the Note below). It must be specified if 'jobname' is not unique within the PUN queue. *Note:* The jobnumber in the punch queue is different from the number of the job which produced the output in either of the following cases: 1. The job stream was submitted via an /INCLUDE with the ICCFSLI option **and** contained a * $$ PUN statement. 2. The job had been run under control of the VSE/POWER PNET networking function.
jobsuffix	is the 1 to 3 digit job suffix which designates the segment number. If this parameter is omitted, segment number 1 is assumed.
jobclass	is the class (A-Z) of the spooled output in the VSE/POWER PUN queue. It must be a class for which no punch unit has been started. The default output class is Q[4].
<u>KEEP</u> |DELETE	specifies the disposition of the output in the VSE/POWER PUN queue after retrieval is complete. KEEP is the default. If DELETE is specified, the job will automatically be deleted from the VSE/POWER LST queue (if the queue entry is segmented and you do not specify a jobsuffix, all segments of the queue entry will be deleted). Abnormal termination results in the output being kept.

MEM=member
MEM=*
MEM=$$PRINT
	specifies the name of the library member into which the punch output retrieved from the VSE/POWER punch queue is to be placed. If 'MEM=*' or nothing is specified, the data is placed in the punch area. $$PRINT causes the punch data to be placed in the print area, which allows you to view the output before deciding whether to place it into a library member.

4 For your installation a different default may have been defined.

PWD=jobpassword	is the 1 to 8 alphanumeric character password that you assigned to the job and its output at submission time (see SUBMIT procedure) or, if you are allowed to submit your own VSE/POWER JECL, the password assigned to the list output according to the rules of VSE/POWER. You can omit it if there is none or if you are the VSE/ICCF administrator. *Note:* Jobs submitted via the card reader are protected by an unprintable password if you do not provide one. Output from such jobs can only be retrieved by the VSE/ICCF administrator.

The GETP procedure lets you do most of your compilations in a batch partition, thus balancing the work load within the system and freeing interactive partitions for interactive processing. The punch queue output can then be retrieved for loading and execution within an interactive partition.

This procedure can also be used for online updating of the VSE/ICCF library file. For example, by using a utility program like OBJMAINT, SYSIN input can be punched to VSE/POWER and then retrieved online by GETP and placed in a library member.

Notes:

1. *Only the submitter or the user designated in the DEST parameter of the VSE/POWER JECL statement * $$ PUN may access the punch output queue entry. An authorized user may specify ANY in the DEST parameter (authorization is established in the VSE/ICCF user profile). This makes the punch output accessible to any user.*

2. *The GETP procedure invokes the DTSGETQ program. The DTSGETQ program runs in an interactive partition and processes the procedure based on the positional parameters 'jobname', 'jobnumber', 'jobsuffix', and 'jobclass'. These are the only positional parameters in the procedure call and they must be specified as shown. They are separated by blanks. The other (keyword) parameters, including their given values, can appear anywhere in the procedure call.*

Examples

1. Place VSE/POWER punch job 'MYJOB' in punch area. The job has no password:

   ```
   getp myjob
   ```

2. Display VSE/POWER punch job 'MYJOB':

   ```
   getp myjob mem=$$print
   ```

GETR Procedure

The GETR (GET READER QUEUE) procedure retrieves a job from the VSE/POWER reader queue and stores it as a member of the VSE/ICCF library file.

```
GETR      jobname [[[jobnumber] [nonnumjclass]]|[jobnumber[ numjclass]]]
                 [KEEP|DELETE] [MEM=member] [PWD=jobpassword]
```

jobname	is the 2 to 8 character name of the job in the VSE/POWER RDR queue you want to retrieve.
jobnumber	is the 1 to 5 digit number assigned to the VSE/POWER job. If omitted, the first job in the VSE/POWER reader queue with the specified jobname will be taken.
[non]numjclass	is either a numeric class (0 - n, where 'n' is the number of the VSE partition) of the spooled input in the VSE/POWER RDR queue, or a nonumeric class (A - Z). A numeric jobclass must never be specified without jobnumber; otherwise the jobclass would be mistaken as the jobnumber. The default class is A.
KEEP |DELETE	specifies the disposition of the VSE/POWER job after retrieval is complete. KEEP is the default, which means that the disposition of the job will not be changed. If DELETE is specified, the job will automatically be deleted from the VSE/POWER reader queue after successful retrieval. Abnormal termination results in the input being kept.
MEM=member	specifies the name of the VSE/ICCF library member into which the VSE/POWER job is to be placed. If no member name is specified, jobname will be used as member name.
PWD=jobpassword	is the 1 to 8 alphanumeric character password that you assigned to the job at submission time (see SUBMIT procedure) or, if you are allowed to submit your own VSE/POWER JECL, the password assigned to the job according to the rules of VSE/POWER. You can omit it if there is no password protection, or if you are the VSE/ICCF administrator. *Note:* Jobs submitted via the card reader are protected by an unprintable password if you do not provide one. Output from such jobs can only be retrieved by the VSE/ICCF administrator.

A normal terminal user may retrieve only jobs that he originated whereas the VSE/ICCF administrator may retrieve any job.

Once it has been retrieved, the job will have no VSE/POWER job or end-of-job statements. Similarly, in place of a * $$ RDR statement, the job will contain the diskette data that are addressed by that statement.

Note: Jobs submitted to another node in a VSE/POWER controlled network cannot be retrieved.

Examples

1. Place the job MYJOB, jobnumber 123, into the VSE/ICCF library member MYJOB. Keep the job after retrieval.

```
getr myjob 123
```

2. Place the job MYJOB, jobnumber 10557, password SECRET, into the VSE/ICCF library member MYMEM, and delete MYJOB from the VSE/POWER reader queue after successful transfer.

```
getr myjob 10557, mem=mymem pwd=secret delete
```

/GROUP Command

The /GROUP command creates a generation member group. It can also be used to remove the group characteristic from the members of the group, and also to reform the members back into a group. The command is effective in command mode.

```
/GRoup      CReate    name [password] [PRIV|PUBL] [n] [DBCS=ON|OFF]
            UNgroup   name [password]
            REGroup   name [password] [DBCS=ON|OFF]
```

CReate	allows you to establish a generation member group within your library.
UNgroup	removes the generation member group characteristic from the members of the group.
REGroup	forms a generation member group from existing members.
name	is the unique 1 to 6 character name for a generation member group, beginning with an alphabetic character.
password	need only be specified if one or all members of the group are to be password protected.
PRIV|PUBL	makes the members of the group either public or private.
n	is a decimal number from 2 to 10 indicating the number of entries to be created in the group. If omitted, 3 is assumed.
DBCS=ON|OFF	sets or resets the DBCS attribute for all members of a generation group. If omitted, DBCS=OFF is assumed.

The /GROUP CREATE command creates from 2 to 10 entries in the library. To the specified names of these entries is added a 2-character suffix in the form '-n'. Whenever data in the input area is saved (/SAVE command), it becomes the '-0' version of the member. The previous '-0' version of the member becomes the '-1' version, the '-1' becomes the '-2' version, and so on. The oldest member in the group is purged.

The /GROUP UNGROUP command returns the members of the group to the same status as ordinary library members. They can thus be handled as such.

All the members to be grouped with the /GROUP REGROUP command must begin with the 1 to 6 character group name followed by the '-n' suffix. The resulting generation member group will consist of all library members beginning with 'name-0' up to and including 'name-n'. 'n' represents the highest suffix in the library. If there is a break in the suffix numbering, only the first continuously suffixed members will be grouped.

The operands PRIV or PUBL can be specified to make the members of your group private or public. If this operand is not specified, the private or public status will be set according to the default in your user profile.

Notes:

1. *If the name of a generation member group ends with '.P', all members in the group are print-type members. It is not possible to have only one print-type member within a generation member group.*

2. *You can change the DBCS attribute for each member of the group separately with the /PROTECT command. In this case, however, it is your responsibility to keep the group consistent.*

Examples:

1. Create a member group consisting of the 4 entries XXX-0, XXX-1, XXX-2, XXX-3 in your library.

```
*READY
/group create xxx 4
*GROUP CREATED
*READY
```

2. Assume that you have a group named XXX whose newest member would be called XXX-0. You want to make some changes to XXX-0 and yet keep the current version. You might do this as follows:

```
*READY
/input
/insert XXX-0
*END INSERT
/end
*READY
/edit
 .  (editor commands to make changes)
quit
END EDIT
*READY
/save XXX-0
*GENERATION MEMBER SAVED
*READY
```

   At the conclusion of this example, the changed version will be called XXX-0 and the previous version will be called XXX-1.

3. The following example shows a group consisting of 4 members (XXX-0 through XXX-3) to which a fifth entry is added.

```
*READY
/group ungroup xxx
*GROUP FUNCTION PERFORMED
/inp
dummy card
/save xxx-4
*SAVED
*READY
/group regroup xxx
*GROUP FUNCTION PERFORMED
*READY
```

[/]HARDCPY Command

The [/]HARDCPY command sets hardcopy mode on or off and directs 3270 terminal output to a hardcopy printer. It directs this output to a private print destination queue for hardcopy output.

```
[/]HARdcpy      devicename
                queuename
                ON
                OFF
                *
                START d q
```

devicename	the logical 4-character name associated with the printer on which the hardcopy is to be produced.
queuename	the logical 4-character name of a private print destination queue.
ON	sets hardcopy mode on after the printer devicename has been entered at least once.
OFF	terminates hardcopy mode.
*	applies only to the remote 3275 with a printer directly attached, and sets continuous output mode.
START	START directs print output to an actual printer associated destination. Enter the [/]HARDCPY command followed by the START operand followed by the printer device name and the private destination name.

This command is only effective for IBM 3270 terminals and cannot be issued from a procedure, because DTSPROCS always assumes an IBM 2741 terminal. The /HARDCPY command (with /) is effective in all modes except edit mode. For normal hardcopy output, specify the /HARDCPY command followed by the 4-character device name of a printer to which you have access.

While in hardcopy mode, all output which would have been displayed will be directed to the hardcopy destination until the /HARDCPY OFF command is given. The data being sent to the printer will also be displayed as verification that the print operation is in progress.

For lengthy output, enter the /HARDCPY command, wait for the first screen of output to appear, press the CLEAR key and then enter the /CONTINU command. The output will then be displayed continuously, and you will not have to press ENTER to receive each new screen of output.

All data directed to printers is queued in disk storage areas, thus allowing you to rapidly dispose of print data. The printing is done when the printer becomes available. Meanwhile you can press the CLEAR key, enter the /HARDCPY OFF command and proceed with other work.

Your installation may have established certain private print destinations not directly associated with actual printers. The VSE/ICCF administrator may make one of these destinations available to you. In this case, set hardcopy mode by entering the command followed by the 4-character destination identifier ('queuename' operand). A private destination reduces the risk of your output being mixed with that of other users.

If you are using an IBM 3275, the [/]HARDCPY * command starts non-disk-queued hardcopy operations back to your printer. That is, each transmission of data normally sent to the display is sent to the attached printer. For queued-on-disk printing, you must specify /HARDCPY followed by some other printer name (not your own printer) or a private destination queue name. You can queue your printout on a private queue. When you have finished terminal operations, enter the /HARDCPY START command followed by the name of your terminal, and the private destination queuename. Printing will continue until the queue is empty.

Notes:

1. *When a printer directed data stream appears on a display terminal, CLEAR must be pressed before you enter VSE/ICCF commands.*
2. *The /CONTINU command handles print hard copy quickly.*
3. *Full screen mode cannot be invoked while hardcopy mode is in effect.*
4. *Before entering a command in hardcopy mode, press the CLEAR key.*
5. *Print-type members are printed using the full width (up to 132 characters) of the hardcopy printer.*
6. *When /LISTP is used in hardcopy mode, all forms control characters are changed to 'write with single space' (X'15'), except for CICS/VS controlled 328x terminal printers with form feed feature. For these a skip to next print page is supported.*
7. *Control characters contained in data sent to the screen are not interpreted in hardcopy mode. Thus, multiple printer lines may occur on one screen line (see example).*

Example:

Step 1: Connect hardcopy printer:

```
/har 186p_
 ...+....1....+....2....+....3....+....4....+....5....+ ..CM
*READY
            *HARDCOPY MODE SET5*READY59_
```

```
      Response is in print format with print control characters (5,9)
      without scale line. Press CLEAR before entering the next command.
```

Step 2: List the member MYMEM:

```
/list mymem_
 ...+....1....+....2....+....3....+....4....+....5....+ .CM

            LINE 15LINE 25LINE 35LINE 45*END PRINT5*READY59_
```

```
      Response is display of member in print format. Press CLEAR key.
```

Step 3: Disconnect hardcopy printer:

```
/har off_
 ...+....1....+....2....+....3....+....4....+....5....+ .CM
*READY
```

```
_
 ...+....1....+....2....+....3....+....4....+....5....+.. .CM
*HARDCOPY MODE NOW OFF
*READY
```

Result: Output on hardcopy printer:

```
*HARDCOPY MODE SET
*READY
LINE 1
LINE 2
LINE 3
LINE 4
*END PRINT
*READY
```

HC Macro (3270 Only)

The HC macro switches an IBM 3270 to hardcopy mode, executes a specified command and returns to normal display mode.

```
HC              command operands
```

command	is any VSE/ICCF system command which is to be executed in hardcopy mode.
operands	are any operands associated with the command specified.

The HC macro executes any given system command and routes the output to a hardcopy printer.

This macro command only applies to 3270 terminals and cannot be invoked in edit mode. Also, a /HARDCPY or HARDCPY command with the printer specified must already have been issued during the session so that the HC macro knows to which hardcopy printer it should direct the output.

If entering list type commands, the continuous version of the command (/LIST) should be used to avoid having to repeatedly press the ENTER key or having to enter the /CONTINU command.

Examples:

1. Print a member on the hardcopy printer:

   ```
   hc /listc xfile
   ```

2. Print a full library display on the hardcopy printer:

   ```
   hc /libc full all
   ```

3. Display status information on the hardcopy printer:

   ```
   hc /status
   ```

HELP Macro

The HELP macro displays summary and detailed information on using VSE/ICCF commands, macros, procedures, and job entry statements. The help information can be presented in any language, including languages that have a double-byte representation like the Japanese Kanji. The HELP macro can only be invoked in command mode.

```
HELP          [statement|/SET [option]|EDITOR|JES|LINE]
```

statement	is the name of a command, macro, procedure, or job entry statement.
/SET [option]	requests HELP information on the /SET command. 'option' is any of the first operands of the /SET command.
EDITOR	requests a display of all editor commands that are explained in the HELP facility.
JES	requests a display of all job entry statements that are explained in the HELP facility.
LINE	requests a display of all editor line commands that are explained in the HELP facility.

If the HELP macro is issued without an operand, an overview panel will be displayed. This panel lists all items for which HELP information is available and tells you how to retrieve that information.

Note: HELP information on rarely used VSE/ICCF functions is not available.

Example:

```
*READY
help
 (HELP information is displayed)
*END PRINT
*READY
```

/INPUT Command

The /INPUT command causes the terminal to leave command mode and enter input mode.

```
/INPut          ┌                              ┐
                │PRompt [INClude|OFF|nn]       │
                │NOprompt                      │
                └                              ┘
```

PRompt	in connection with input verification set off (see /SET VERIFY), causes a prompt number to appear on the terminal identifying the next line to be entered.
NOprompt	eliminates any type of prompting even if your user profile indicates automatic prompting.
INClude	causes the prompt number to become part of the line entered.
OFF	eliminates any type of prompting, even if your user profile indicates automatic prompting.
nn	is a decimal number from 1 to 32767 to be used as prompt increment. The default is 10.

After the /INPUT command has been accepted, all lines that are not system commands recognizable by VSE/ICCF are placed in your input area.

The /INPUT command is only effective in command mode. It places the terminal into system input mode. Do not confuse this input mode with the editor input mode. Editor input mode allows any type of data (including VSE/ICCF commands) to be entered into the input area; input mode does not allow VSE/ICCF system commands to be entered into the input area.

The /INPUT command does not **immediately** destroy the previous contents of the input area. However, after the first line of input has been entered, the previous contents of the input area will have been destroyed.

Once the /INPUT command has been entered, you can type in any number of input lines (job entry statements, programs, data, etc.). The only limit is the maximum size of your input area as defined in your user profile. To terminate entry of input records, issue the /END command.

In addition to typing in data, you can copy all or part of a library member into the input area using the /INSERT command.

If you enter PROMPT with no further operands or if your user profile indicates automatic prompting, and input verification is set off, VSE/ICCF displays the last line entered and shows the line number of the next line to be entered. The prompt number is not added to the data you enter. If you are using input verification, a maximum of the last ten lines is displayed and no prompt number will appear on your screen.

If PROMPT with either INCLUDE or 'nn' is specified, inclusion prompting will be in effect. Inclusion prompting causes the prompt number to actually become part of the line entered. When the INCLUDE operand is specified, the first prompt will be the line number times 10. This 5-digit number will occupy columns 1 through 5 of each input line. When inclusion prompting is set by specifying the 'nn' operand, 'nn' will be used as the prompting increment instead of the default increment of '10'. If your user profile indicates automatic inclusion prompting, you need not enter the INCLUDE operand.

Notes:

1. *To erase the contents of your input area, issue the /INPUT command followed by the /CANCEL command.*

2. *If you require more space than is in your input area, save the contents of the input area (/SAVE command) and start a new one. (/INPUT command). After the new area has been saved, the multiple areas can be logically connected using the /INCLUDE facility.*

3. *Inclusion prompting (INCLUDE or 'nn' operand) is especially useful when entering VS BASIC programs which require a line number at the beginning of each statement. This type of prompting saves you from having to type the statement number. (For prompting with sequence numbers in other than columns 1 to 5, see "Linenumber Editing" on page 4-31.)*

4. *See the /PROMPT command for an explanation of how to vary prompting characteristics during an input session.*

5. *When inclusion prompting is set on, the included prompt disregards lines beginning with a slash. Job entry statements can thus be entered even though inclusion prompting is set on.*

6. *The /INPUT command cannot be used while an asynchronous (see /ASYNCH command) interactive partition execution is in progress, since the input area is the job stream being run. However, you can retain a dummy library member into which you can edit or insert data while in asynchronous mode.*

Example 1:

```
*READY
/input
111
222
/end
*READY
/save group1
*SAVED
*READY
/input
333
444
/insert group1
*END INSERT
555
/end
*READY
/list
333
444
111
222
555
*END PRINT
*READY
```

Example 2:

```
*READY
/input prompt include
/load basic  .
input j,k    .  (The prompt will be typed back for you)
l = j + k    .
print l      .
end          .
/end
*READY
/list
/LOAD BASIC
00020 INPUT J,K
00030 L = J + K
00040 PRINT L
00050 END
*END PRINT
*READY
```

/INSERT Command

The /INSERT command copies all or part of a library member, or the print, punch or log area, into the input area. If the library member was in compressed format, it is also decompressed. Effective in input mode.

```
 /INSert        name [password] [m [n]]
```

name	is the name of a library member which is to be copied into the input area. This member must exist in your primary, connected or common library. 'name' can also be $$PUNCH, $$PRINT or $$LOG for the punch, print or log areas. If 'name' refers to a DBCS member, the double-byte characters are not displayed correctly. However, if you save the new member and set the DBCS attribute for it (/PROTECT), the double-byte characters will be displayed in their correct form.
password	need only be specified if the member is password protected.
m	is the first line to be inserted. If 'm' is not specified, line 1 is assumed. For compressed members line 1 is assumed, regardless of what has been specified. The maximum value is 99999.
n	is the last line to be inserted. If 'n' is not specified, end-of-file is assumed. The maximum value is 99999.

The copy operation terminates if it reaches the limit of the input file specified in your user profile. To copy the remaining records, save the contents of the input area and issue a second /INPUT command.

To copy only a portion of a member into the input area, specify a starting and ending line number (use the /DISPLAY command to find these numbers).

Specifying $$PUNCH, $$PRINT or $$LOG allows you to save the output of compilers or other utilities. By copying the contents of the punch area with the /INSERT command and then saving the input area, you can save an object program produced by a language compiler. You can also save the contents of the print or log areas for future reference.

The /INSERT command offers the standard way of decompressing a compressed member.

Example:

Please see example 1 of the /INPUT command for an example of this command.

/LIBC Command (See [/]LIBRARY Command)

[/]LIBRARY Command

The [/]LIBRARY command displays directory information for individual libraries to which you have access.

```
[/]LIBrary    ┌       ┐  ┌          ┐
/LIBC         │CONn   │  │FULL ┌   ┐│
              │COMmon │  │*xxx │ALL││
              └       ┘  └     └   ┘┘
```

CONn COMmon	selects the library for which you want directory information displayed. CONn stands for your connected library. COMmon stands for the common library. If omitted, the directory information for your primary library is displayed (see Example 1).
FULL[ALL]	displays the entire contents (see Example 2) of directory information, that is: not only the names. ALL causes directory information for all members to be displayed, not only the members that you own (ALL is assumed if in your user profile OPTB bits 2, 3, or 6 are on; these are essentially VSE/ICCF administrator bits).
*xxx [ALL]	displays the FULL directory information for members that have names starting with 'xxx' (see Example 3). 'xxx' can be a string from 1 to 7 characters. ALL causes directory information for all members to be displayed, not only the members that you own (ALL is assumed if in your user profile OPTB bits 2, 3, or 6 are on).

/LIBC causes the directory information to be scrolled continuously to the end. This is only useful if the display needs more than one screen.

When neither FULL nor *xxx are specified, a short form of the directory information is displayed (see Example 1).

The FULL listing is sequenced by location within the directory. The directory is maintained on a last-in/first-out basis, so that the recently added members usually appear at the beginning of the directory.

Notes:

1. *Use the SDSERV procedure to obtain a sorted directory listing.*
2. *The /LIB command does not use the last two lines of your screen. These are reserved for messages.*
3. *The date can appear in the form Month, Day, Year (MDY), or Day, Month, Year (DMY), depending on the format active when VSE/ICCF was started.*

Examples:

1. You ask for a short display of the member names in your primary library.

```
/lib_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...CM
*READY
```

```
_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...CM
LIBRARY NO. - 00005   01/27/83  17/14/44   DIR ENTRIES: ACT=00009 MAX=00200

JAC005    ********  DTSANAL1  SCALE     GROUP1    FORTPROG
LETTERS   DIGITS    FORTDTA
*END PRINT
*READY
```

Member names are displayed in directory sequence. '********' entries represent empty directory slots, created either because the member was purged or free space was requested when the library was restored.

2. Display full information of the members owned by you contained in your connected library.

```
/lib con full_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...CM
*READY
```

```
_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...CM
LIBRARY NO. - 00013   02/01/83  13/54/03   DIR ENTRIES: ACT=00012 MAX=00200

0003 HELLO     09/13/82 COOL
0006 PAYROLL  P08/14/82 COOL PRIV CPR
0009 SALES-0   08/14/82 COOL          GDG
0010 SALES-1   08/14/82 COOL          GDG FL
*END PRINT
*READY
```

1st column:	'nnnn'	Directory entry sequence number.
2nd column:	'xxxxxxxx'	Member name.
3rd column:	'P'	Password protection indication .
4th column:	'mm/dd/yy'	Creation or replacement date.
5th column:	'uuuu'	Member owner's userid ('MSG*' if member is a message member)
6th column:	'PRIV'	Indication for a private member.
	' '	Indication for a public member.
7th column:	'CPR'	Compressed member indicator.
8th column:	'GDG'	Group member indicator. Indicates that the member is part of a group.
9th column:	'FL'	Editor change flagging indicator.

3. Display full information of the members owned by you contained in your primary library and starting with 'F'.

```
/lib *f_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...CM
*READY
```

```
-
 ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...CM
LIBRARY NO. - 00005   01/27/83  17/14/44   DIR ENTRIES: ACT=00009 MAX=00200

0006 FORTPROG  02/14/79 COOL PRIV
0009 FORTDTA   02/14/79 COOL PRIV
*END PRINT
*READY
```

LIBRC Macro

The LIBRC macro is used to catalog a VSE/ICCF library member or the punch area into a VSE sublibrary.

```
LIBRC   l.s mn.mt {$$PUNCH|membername [password]} [REPLACE] [DATA=YES][EOD=xx]
```

l	is the VSE library name
s	is the VSE sublibrary name
mn	is the VSE member name
mt	is the VSE member type
$$PUNCH	denotes the punch area whose contents is to be cataloged into the specified VSE sublibrary.
membername	is the name of the member in the VSE/ICCF library file.
password	is the password of the VSE/ICCF library member if the member is password protected.
REPLACE	indicates that if a member with the specified name already exists in the VSE sublibrary, the member should be replaced by the new data.
DATA=YES	This parameter applies only to cataloging a VSE **procedure**. It indicates that the procedure contains SYSIPT data.
EOD=xx	specifies two end-of-data characters. If omitted, the end-of-data delimiters are assumed as '/+'. Defining delimiters other than '/+' is particularly useful if you catalog a VSE procedure which contains '/+' as data in columns 1 and 2.

If you invoke the macro with less than two operands, the macro displays the complete invocation syntax.

Note: The LIBRC macro requires an interactive partition size of 256K.

Example:

Catalog the VSE/ICCF library member DHTA1 into the VSE sublibrary PRVICCF.CICS with the member specification NAME1.TYPE1. A member with this name and type already exists in the target library.

```
librc prviccf.cics name1.type1 dhta1 replace_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+. .CM
*READY
```

```
_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+. .CM
*RUN REQUEST SCHEDULED FOR CLASS=A
* * * * * START OF PROCEDURE (CLIST) * * * * *
  ACCESS S=PRVICCF.CICS
L113I RETURN CODE OF ACCESS IS  0
  CATALOG NAME1.TYPE1          EOD=/+  REP=Y
L113I RETURN CODE OF CATALOG IS  0
*PARTIAL END PRINT
**** JOB TERMINATED - ICCF RC 00, EXEC RC 0000, NORMAL EOJ
*READY
```

LIBRL Macro

The LIBRL macro displays a member of a VSE sublibrary, or stores it in print-type format as a member in your VSE/ICCF library or in the print area.

```
LIBRL    l.s mn.mt [$$PRINT|membername [password]] [REPLACE]
```

l	is the VSE library name
s	is the VSE sublibrary name
mn	is the VSE member name
mt	is the VSE member type
$$PRINT	denotes the print area, where member mn.mt is to be stored in print-type format.
membername	is the name of the VSE/ICCF library member into which member mn.mt is to be stored. Because the member is stored in print-type format, the two rightmost characters should be '.P' to declare the member as a print-type member. If 'membername' is not specified, $$PRINT is taken as the default.
password	is the password of the VSE/ICCF library member if the member is password protected.
REPLACE	indicates that if a member with the specified name already exists in the VSE/ICCF library file, the member should be replaced by the new data.

If you invoke the macro with less than two operands, the macro displays the complete invocation syntax.

Note: The LIBRL macro requires an interactive partition size of 256K.

Example:

Display the member NAME1.TYPE1 of the VSE sublibrary PRVICCF.CICS.

```
librl prviccf.cics name1.type1 $$print
 ...+....1....+....2....+....3....+....4....+....5....+....6....+. .CM
*READY
```

```
_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+. .CM
*RUN REQUEST SCHEDULED FOR CLASS=A
  ACCESS S=PRVICCF.CICS
L113I RETURN CODE OF ACCESS IS  0
  LIST NAME1.TYPE1
MEMBER=NAME1.TYPE1     SUBLIBRARY=PRVICCF.CICS    DATE: 84-01-17
                                                  TIME: 14:10
----------------------------------------------------------------
   .
   .      (member data)
   .
L113I RETURN CODE OF LIST IS 0
*PARTIAL END PRINT
DATA STORED INTO ICCF-MEMBER OR $$PRINT AREA
*READY
```

LIBRP Macro

The LIBRP macro punches a member from a VSE sublibrary, or stores it as a member in your VSE/ICCF library or in the punch area.

```
LIBRP       l.s mn.mt [$$PUNCH|membername [password]] [REPLACE]
```

l	is the VSE library name
s	is the VSE sublibrary name
mn	is the VSE member name
mt	is the VSE member type
$$PUNCH	denotes the punch area, where member mn.mt is to be stored.
membername	is the name of the VSE/ICCF library member into which member mn.mt is to be stored. If 'membername' is not specified, $$PUNCH is taken as the default.
password	is the password of the VSE/ICCF library member if the member is password protected.
REPLACE	indicates that if a member with the specified name already exists in the VSE/ICCF library file, the member should be replaced by the new data.

If you invoke the macro with less than two operands, the macro displays the complete invocation syntax.

Note: The LIBRP macro requires an interactive partition size of 256K.

Example:

Have the member NAME1.TYPE1 of the VSE sublibrary PRVICCF.CICS punched and stored as the VSE/ICCF library member DATA1.

```
librp prviccf.cics name1.type1 data1 replace_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+. .CM
*READY
```

```
_
 ...+....1....+....2....+....3....+....4....+....5....+....6....+. .CM
*RUN REQUEST SCHEDULED FOR CLASS=A
  ACCESS S=PRVICCF.CICS
L113I RETURN CODE OF ACCESS IS  0
  PUNCH NAME1.TYPE1
L113I RETURN CODE OF PUNCH IS 0
*PARTIAL END PRINT
DATA STORED IN MEMBER DATA1
*READY
```

/LIST Command

The /LIST command displays the contents of the input, punch, print or log area, or of a library member. It is valid in command and input mode.

```
/List          ┌[m [n]] [name [password]]┐
/LISTC         │  *                      │
/LISTX         └                         ┘
```

/LISTC	displays data in continuous output mode (as if the /CONTINU command had been entered).
/LISTX	displays data in combined character and hexadecimal format.
m	is the first line number (1 to 99999) to be displayed. If m is omitted, 1 is assumed.
n	is the last line number (1 to 99999) to be displayed. If n is omitted, the display continues through end of file.
*	displays the last ten lines of the input area from your current position.
name	is the name of the library member to be displayed. If omitted, the input area is displayed. Name can be $$PUNCH, $$PRINT or $$LOG, if the contents of these areas are to be displayed. If 'name' refers to a DBCS member or to the print area, the data to be displayed can also be mixed data.
password	need be specified only if the member is password protected.

To display a portion of the file, specify a starting and ending line number. If the ending line number ('n') exceeds the number of lines in the file, the display proceeds to the end of the member. The library member need not be decompressed to be displayed.

3270 users can press the Display key [5] to display the last ten lines in the input area.

While you are displaying a file, your terminal is in list mode until end-of-file is reached. If the listing needs more than one screen, you can scroll through the file in several ways: press ENTER, issue the /SKIP command, or use your PF keys. With /CONTINU you can start or stop continuous display, and with /HARDCPY you can transfer the output to a hardcopy printer. To leave list mode, enter the /CANCEL command. Or, if you are using a 3270, press the Cancel key[6]. Your terminal reenters the mode it was in when you entered the /LIST command.

For a print-type member, the first two bytes of each record are interpreted as print control characters, and the member is displayed in printline format. The default PF key settings (as described in Figure 2-1 on page 2-12) are **not active**; you can use your own PF key settings, instead.

5 The definition of the Display/Attention key is a VSE/ICCF tailoring option. The default is PA3. It may be different for your system.

6 The definition of the Cancel key is a VSE/ICCF tailoring option. The default is PA2. It may be different for your system.

Notes:

1. *The /DISPLAY and /LIST commands are the same, except that the /DISPLAY command also prints line numbers and is only valid for non-compressed members.*

2. *Unless the default line size is more than 72 (see /SET command), sequence numbers in columns 73 to 80 are not displayed, whereas other data are.*

3. *When the contents of your log area are displayed (/LIST $$LOG), the entries appear in reverse order. That is, the most recent entries appear first.*

4. *The /LIST command does not use the last two lines of your screen. These are reserved for messages.*

5. *Print-type members are displayed on the IBM 3278 Model 5 wide screen using up to 132 display positions per line.*

6. *Print-type members are printed on a hardcopy printer using the total line size.*

7. *The records in a print-type member that do not contain valid print control characters in the first two columns are printed as 80-byte records.*

8. *If LISTX is specified for a print-type member, the member will be displayed in 80-character record format, not in print format.*

9. *The second ENTER after issuing the /SKIP BOTTOM or /SKIP END command terminates the /LIST command.*

Example:

```
*READY
/input
first line
second line
/list
FIRST LINE
SECOND LINE
*END PRINT
third line
/end
*READY
/save sampl1
*SAVED
*READY
/l sampl1
FIRST LINE
SECOND LINE
THIRD LINE
*END PRINT
*READY
```

/LISTC Command (See /LIST Command)

/LISTP Command

Displays print output from the VSE/POWER list queue. It can only be used in command mode.

```
/LISTP      [jobname [jobnumber [jobsuffix]] [jobclass] [PWD=password]]
/LP
```

jobname	is the name of a job previously submitted which now has output in the VSE/POWER list queue. If this parameter is omitted, jobname is assumed to be the 8-character name formed by the userid and the terminal identification.
jobnumber	is the 1 to 5 digit jobnumber assigned to the VSE/POWER list output. Specify the jobnumber if the jobname is not unique within the VSE/POWER list queue.
jobclass	is a single alphabetic character specifying the output class associated with the job. This operand needs only be specified if your list output does not have the default output class Q[7].
jobsuffix	is the 1 to 3 digit job suffix which designates the segment number. If this parameter is omitted, segment number 1 is assumed.

PWD=password specifies the 1 to 8 alphanumeric password assigned to the VSE/POWER list output.

The data to be displayed can be alphanumeric or mixed data. Note, however, that data which has already been prepared for printing on a 3200 printer, for example with Utility PRPQ (5799 BGF), is not displayed correctly any more. If you request a job, identified by jobname and possibly jobnumber, to be displayed and this jobname (and jobnumber) occurs more than once in the queue, the job occurring first will be displayed.

This command would normally be used to display the print output from a job previously submitted to VSE/POWER to be run in another VSE partition. If the SUBMIT procedure was used, the jobname and also the jobnumber assigned to it by VSE/POWER would be returned to you. This name is either the name of the library member submitted or the name formed by your userid and the terminal identification.

The execution of the job must have been completed before you can display the print output. To check the status of the job, issue the /STATUSP or the /DQ command. Or you can wait for VSE/POWER to send you the completion message.

While displaying the print output, your terminal is in list mode until end of output is reached. If the display needs more than one screen, scroll through the file by pressing ENTER, or by using the /SKIP command. Use the /SHIFT command to 'scroll' left and right. With /LOCP you can locate a character string in the print output. With /CONTINU you can start or stop continuous display, and

[7] For your installation a different default may have been defined.

with /HARDCPY you can transfer the output to a hardcopy printer. You can leave list mode at any time with the /CANCEL command. Or, if you are using a 3270 you can press the Cancel key[8].

Output from the VSE/POWER list queue is displayed in printline format, that is, part of the print line may be invisible (this is contrary to 'wrap-around' or 80-character format where two lines are displayed for one print line). The /SHIFT command allows you to move the screen window over to the portion that is invisible. The default PF key settings (see Figure 2-1 on page 2-12) are **not active**; use your own PF key settings, instead. If you want to use those default PF key settings, you must issue first the command /SHIFT OFF.

You can decide yourself how to dispose of the output. For example, you could erase it using the /ERASEP command. Or, you could use the /ROUTEP command to route it to the installation printer, or to a VSE/POWER RJE printer. The /ROUTEP command changes the disposition of the file to the default of DISPATCHABLE (D)[9].

Notes:

1. *Certain jobnames (for example, ALL) should be avoided; refer to the section "Submit-to-Batch Capability" on page 9-29.*

2. *If /LISTP is used in hardcopy mode, forms control characters are changed to 'WRITE WITH SINGLE SPACE' (X'15'), except for CICS/VS controlled 328x terminal printers with the form feed feature. For these a skip to next print page is supported.*

3. *If jobname, jobnumber, jobsuffix and jobclass do not identify a unique print output, the first occurrence is taken.*

4. *Only the submitter or the user designated in the DEST parameter of the VSE/POWER JECL statement * $$ LST may access the list output queue entry. An authorized user may specify ANY in the DEST parameter (authorization is established in the VSE/ICCF user profile). This makes the list output accessible to any user.*

5. *The second ENTER after issuing the /SKIP BOTTOM or END command terminates /LISTP and returns you to command mode.*

Examples:

1. List the VSE/POWER output from 'MYJOB', which has the jobnumber 0200 and the password 'SECRET' in the installation default class:

```
/listp myjob 0200 pwd=secret
```

2. Same as above but with the output class 'A':

```
/listp myjob 0200 a pwd=secret
```

[8] The definition of the Cancel key is a VSE/ICCF tailoring option. The default is PA2. It may be different for your system.

[9] For your installation a different default may have been defined.

LISTX Command (see /LIST Command)

LOAD Procedure

The LOAD procedure causes an object program to be loaded into storage and run.

```
LOAD          [name1|*] [JES name2|*] [DATA name3|*] [options]
```

name1	is the name of the library member containing the object deck. If this operand is '*', the object module will be read from the punch area.
JES name2	is the name of a member containing the file definitions and other job entry statements for the running of the program. '*' indicates that you are to be prompted for the job entry statements. To end prompting hit the ENTER key.
DATA name3	is the name of the member containing the job stream data for the execution. If '*' is specified, the data will be read conversationally from the terminal as if /DATA INCON had been specified. To end data reading type '/*'.
options	are one or more VSE/ICCF /OPTION statement options to be applied to the execution.

The object program can be in the library (name1 parameter) or in the punch area ('*'). If you have pre-established file definitions and job stream data as library members, you can include these as the 'name2' and 'name3' parameters. If one or both of these is specified as '*', the terminal will be prompted for entry of the job entry statements or data at the appropriate point.

The LOAD procedure is listed in "Examples of Procedures" on page 7-28.

Examples:

1. Load from the punch area and read data from the terminal:

   ```
   load * data * getvis=48K
   ```

2. Load from a library member with JES from another member:

   ```
   load myjob jes fildef
   ```

/LOCP Command

The LOCP command locates a character string in VSE/POWER lists. It can be given in list mode after the /LISTP command.

```
/LOCP          [m|1[ n|156]] string|/string/
```

m — is a number between 1 and 156 specifying the column where the search is to start. If 'm' is not specified, 1 is assumed as the default.

n — is a number between 1 and 156 specifying the column where the search is to end. The string must be **entirely** contained in the zone from 'm' to 'n' if the locate is to be successful. If you specified, for example,

```
/locp 1 100 AAA
```

and in a given record the string AAA occurred in columns 99 through 101, the string would not be located. If 'n' is not specified, 156 or the end of the print line, whatever is smaller, is assumed as the default.

string — specifies the character string to be located.

/string/ — specifies the character string including imbedded blanks that is to be located (/ is the string delimiter character).

The search for a match of the given string starts from the line following the current position. If found, the line with the matching characters is displayed at the top of the screen, below the scale line. The line is shifted so that the data found can be seen on the screen. If not found, processing is repositioned to the point where the command was entered and the message: *CHARACTER STRING NOT FOUND, PRESS ENTER TO RESUME is displayed.

Notes:

1. *To avoid overloading your library unnecessarily with large VSE/POWER files, first check them using the /LISTP and /LOCP commands.*
2. *The date can appear in the form Day, Month, Year (DMY), or Month, Day, Year (MDY), depending on the format that was active when VSE/ICCF was started.*
3. *If a /COMPRESS is in effect for the current display, /LOCP locates the string at the "real" position, not at the position as displayed on the screen. Therefore, a /LOCP after a /COMPRESS may not be practical in all cases.*

Examples:

1. Scan forward from the current line + 1 for the character string **ERROR. The search should be limited to columns 61 up through the end of the line.

```
/locp 61 **ERROR
```

2. Scan forward from current line + 1 for character string /NUMBER OF STATEMENTS/

```
/locp /NUMBER OF STATEMENTS/
```

/LOGOFF Command

The /LOGOFF command ends a session and causes all accounting statistics to be updated.

```
/LOGOFF
```

The /LOGOFF command is effective in input and command mode.

If you have more than one userid (for example, to access more than one library or set of defaults) you can log on with the other userid after first logging off.

The system can log you off in certain situations, for example when you have not used your terminal beyond the time-out limit set for you (see the /SHOW command on how to display your terminal timeout value).

Notes:

1. *The date can appear in the form Day, Month, Year (DMY), or Month, Day, Year (MDY), depending on the format that was active when VSE/ICCF was started.*

2. *Do not forget to log off before leaving the terminal; otherwise you might be charged for terminal time that you did not actually use.*

Example:

```
*READY
/logoff
*DATE=09/02/83 TIME=01/35/30
*YOU ARE NOW LOGGED OFF
*GOOD-BYE FOR NOW
```

/LOGON Command

The /LOGON command starts a terminal session.

```
/LOGON       userid
```

userid is the 4-character user identification.

This command is only effective after 'iccf' (the VSE/ICCF terminal control transaction) has been entered. It is not effective in any other VSE/ICCF modes (input, command mode, etc.).

Example:

```
iccf    (enter VSE/ICCF transaction identification)
*VSE/INTERACTIVE COMPUTING AND CONTROL FACILITY
*PLEASE ENTER: /LOGON WITH YOUR USERID
/logon usrc
*ENTER PASSWORD
@@@@@@
*LOGON COMPLETE - DATE 09/02/83 TIME 01:30
*READY
  .
  .
  .
  (user's session)
  .
  .
/logoff
*DATE=09/02/83 TIME=01/50/00
*YOU ARE NOW LOGGED OFF
*GOOD-BYE FOR NOW
```

/LP Command (See /LISTP Command)

/MAIL Command

The /MAIL command displays messages from either the system operator or the VSE/ICCF administrator.

```
/MAil
```

This command is only effective in command mode.

Messages can be directed to all users (general mail) or to a specific user (specific mail).

| The messages can contain double-byte characters, if the DBCS attribute has been set for the A$MAIL
| member.

Generally, you should display your mail as soon as you have logged on.

Example:

```
*READY
/mail
* * * DISPLAY OF MAIL FILE * * *
*
A NEW PROCEDURE HAS BEEN ADDED TO THE SYSTEM
THIS PROCEDURE (C$SORT) PROVIDES A GENERALIZED
SORT FACILITY.
SPECIFICATIONS FOR USE ARE - - -
 (etc.)
USRA ALL STUDENTS IN INTRO. TO COBOL (CC5738-02) SHOULD
USRA DELETE SAVED MEMBERS FROM THE LIBRARY.
*END PRINT
*READY
```

[/]MSG Command

The [/]MSG command displays (alphanumeric) messages that arrived for you while your terminal is set to non-automatic message display (see the /SET MSGAUTO command under "Set VSE/ICCF Features" on page 3-115 in this chapter).

```
[/]MSG
```

/MSG (with '/') is valid in command, list and execution modes. MSG (without '/') is valid in edit mode.

Messages can be sent by all VSE/ICCF users, the system operator, and VSE/POWER.

Example:

The example shows a job completion message. VSE/POWER notifies you with such a message that the execution of job JOBX submitted by you to VSE/POWER has finished.

```
/msg_
 ...+....1....+....2....+....3....+....4....+....5....+ .CM
*READY
```

```
_
 ...+....1....+....2....+....3....+....4....+....5....+.... ..CM
 03/15 - 13:45 MSG FROM VSE/POWER
1Q5DI EXECUTION COMPLETED FOR JOBX 00135 ON NODE1, TIME=13:44:36
*END MSG
*READY
```

MVLIB Procedure (see CPYLIB Procedure)

/PASSWRD Command

The /PASSWRD command is used to specify a new logon password.

```
/PASswrd     new password
```

new password is a 3 to 6 character word which is to be the new logon password.

You can only change your password if your user profile allows it.

This command is effective only in command mode.

Notes:

1. *Do not forget to write down your new logon password. There is no handy way of obtaining it, should you forget it.*
2. *The password specified as the operand of this command can be specified on the hardcopy printer (2740 and 3767) terminals. To ensure security, you may want to tear off this portion of the printout and destroy it.*

Example:

```
*READY
/passwrd jacpas
*PASSWORD CHANGED
*READY
```

[/] PFnn Command

The /PFnn command invokes a function associated with a PF key. It is valid in any mode.

```
/PFnn        [data]
```

nn is the number of the program function invoked (a decimal number from 1 to 24).

data is a character or variable parameter to be appended to the program function setting.

This facility allows terminals which do not have PF keys to nevertheless allow program key functions. You can assign functions to particular PF keys using the /SET PF command.

The /PFnn command can also be useful on terminals which **do** have PF keys. An example is when you are entering multiple lines separated with the logical line end character; you can imbed a PF key function within your string of commands. Thus, you do not have to press the program function key itself, which avoids a terminal interrupt.

If the program function has been set to include a variable parameter (see /SET PF command), the operand portion of the /PFnn command is substituted for the variable parameter. If the program function has not been set with a variable parameter, any operand specified with the /PFnn command is appended to the end of the program function before it is performed.

Which function do you get when you enter the /PFnn command? You may have defined up to 4 different functions for one PF key: one for command mode (PFnn), one for edit mode (PFnnED), one for execution and conversational read mode (PFnnEX) and one for list and spool mode (PFnnLS). Additionally there exists a default function for each PF key. You will get the function which is defined for the mode you are in; so, if you are in list mode (LS), /PFnn would get you the function assigned to PFnnLS. However if PFnnLS was not defined or if all settings belonging to list mode were suspended (/SET PFLS OFF), you would get the function out of the next deeper level in hierarchy which is PFnn defined for command mode (see figure below), and if this is still not defined you get the default setting.

/PFnn in	Invokes the Function
CM mode	PFnn
ED,FS mode	PFnnED ―> PFnn
EX,RD mode	PFnnEX ―> PFnn
LS,SP mode	PFnnLS ―> PFnn ―> default PF key setting (see Fig.2–1)

Figure 3-2. Program Function Hierarchy

Note: PF keys cannot be used in procedures.

Examples:

```
*READY
/set pf1 /list 1 18 $$LOG
*PROGRAM FUNCTION SET
*READY
/pf1
   .
   . (display of first 18 lines of your log area)
   .
*END PRINT
*READY
/set pf2 /statusp (blank char. must follow 'statusp')
*PROGRAM FUNCTION SET
*READY
/pf2 joba
*STATUS = LL - JOB COMPLETED
*READY
```

PLI Procedure

The PLI procedure causes a library member to be processed by the DOS/VS PL/I optimizing compiler.

```
PLI          name1 [OBJ name2|OBJ *] [PROcess] [options]
```

name1	is the name of a member in the library containing the PL/I language source program to be compiled.
OBJ name2	is to be the library member name of the object module resulting from the compilation. This member need not initially exist.
options	are one or more /OPTION job entry statement options that alter the way the compiler is processed (e.g., NODECK, LIST, etc.).

The PROCESS operand specifies that you are to be prompted for entry of a PLI *PROCESS OPTION statement.

The object module will be placed in your punch area if the OBJ operand is omitted or if OBJ * is specified. If the OBJ operand specifies a member name ('name2' parameter), the object module will be saved in the library under this name. If the named member already exists, the object module will overlay it.

The 'options' operand can be specified as one or more options which influence the compilation, such as XREF, LIST, SYM, etc. If the NODECK option is specified, no object module will be produced even if the OBJ operand is present. The NORESET option should be specified if you are stacking multiple object decks in the punch area.

The LOAD procedure can be used to load and execute the object module produced by this procedure.

Examples:

```
1.  PLI TSTPLI
    LOAD

2.  PLI XXPROG OBJ OBJMEM LIST
    LOAD OBJMEM JES FILDEF DATA *
```

PRINT Macro (3270 Only)

The PRINT macro routes the contents of a library member, or the input area, to a hardcopy printer associated with a 3270 terminal for printing.

```
PRINT        [name [pass]]
```

name	is the name of the library member to be printed. If it is omitted, the input area will be printed.
pass	is the password associated with the member, if any.

A /HARDCPY command that specifies the name of the hardcopy printer to which the data is to be sent must already have been issued during the session.

Print-type members will be printed in print format using up to 132 characters of the hardcopy printer.

/PROMPT Command

The /PROMPT command turns line number or inclusion prompting on or off. It is valid in input mode only.

```
/PROmpt        [OFF|INClude|nn]
```

OFF	prompting is turned off.
INClude	inclusion prompting is set on.
nn	is the decimal increment (1 to 32767) for inclusion prompting, which is also set on.

Line number prompting, in connection with input verification set off (see /SET VERIFY), means that you are prompted for each input line with the number of the next line to be entered. The prompt number is not included in the input data. If you are using input verification, a maximum of the last ten lines is displayed and no prompt number will appear on your screen.

Inclusion prompting means that you are prompted with a five digit number that is included in columns 1 to 5 of your input data. The prompt increment is increased by 10 for each new line, unless you vary it with 'nn'.

Line number prompting will be turned on, if (1) no operand is specified, (2) no prompting is in effect, and (3) input verification is set off. If no operand is specified and line number or inclusion prompting is in effect, the prompting status will not be changed, but the current prompt increment will be displayed.

The /PROMPT command displays the current line number, if this has been lost while printing (/LIST or /DISPLAY) in input mode. It can also be used to change the default in your user profile.

Example:

```
*READY
/set verify off
/input
/prompt
0001 first input line
0002 second input line
/prompt off
third input line
/end
*READY
/list
FIRST INPUT LINE
SECOND INPUT LINE
THIRD INPUT LINE
*END PRINT
*READY
/input
line 1
/PROMPT INC
00020 line 2
00030 line 3
/end
*READY
/list
LINE 1
00020 LINE 2
00030 LINE 3
*END PRINT
*READY
```

/PROTECT Command

The /PROTECT command mainly allows you to control security aspects of members in your VSE/ICCF libraries. It lets you remove a password from a member, change its attribute from PUBLIC to PRIVATE, or vice versa; you can add or remove automatic editor change flagging and change your identification associated with a member. The /PROTECT command can also be used to turn on the compressed member attribute, to clear the update-in-progress (UPIP) attribute, or to set the DBCS attribute on for a member.

```
/PROTect     name [oldpass]  PRIV|PUBL
                             DATE
                             FLAG|OFFL
                             USERID newid
                             NEWPASS newpass
                             NOPASS
                             CPRS
                             UPIP
                             DBCS=ON|OFF
```

name	is the name of the library member whose protection characteristics are to be altered.
oldpass	is the existing four-character password of a password protected member; it has to be entered if the existing member is password protected.
PRIV|PUBL	makes the member either PRIVATE or PUBLIC.
DATE	indicates that the current date is to replace the entry date in the directory record for the member.
FLAG|OFFL	sets automatic editor change flagging on or off for the member. (The OFFL operand can only be used by the VSE/ICCF administrator.)
USERID newid	is the four-character user identification to be applied to the member.
NEWPASS newpass	is the four-character password to be applied to the named library member.
NOPASS	removes password protection from the member.
CPRS	sets the compressed member attribute on.
UPIP	sets the update-in-progress attribute off (can only be used by the VSE/ICCF administrator).
DBCS=ON|OFF	sets or resets the DBCS attribute for the specified member. By default, a newly created member does not have the DBCS attribute.

This command is only effective in command mode.

To add password protection to a member, specify as operands the member name and the password (/PROTECT name NEWPASS newpass).

To remove password protection, specify as operands the member name, the old password and the keyword NOPASS (/PROTECT name oldpass NOPASS).

To change the password of a member, specify the member name, the old password and the new password (/PROTECT name oldpass NEWPASS newpass).

To change a non-password protected member's privacy attribute, specify the member name and either PRIV or PUBL (/PROTECT name PRIV|PUBL).

To change a password protected member's privacy attribute, specify the member name, the password and either PRIV or PUBL (/PROTECT name oldpass PRIV|PUBL).

To apply the date to a member, use the DATE keyword. This causes today's date to replace the previous date in the directory entry for the member. This can be useful if your installation periodically purges old library members based on the date in the directory record.

To add or remove automatic editor change flagging from a member, specify the member name and optional password followed by either FLAG or OFFL (/PROTECT name FLAG|OFFL). This flagging causes the editor to place information in each changed record (usually in columns 73-80) giving the type of change, date of change, and userid of the user making the change. Do not use editor change flagging for RPG source members.

To change the user identification associated with a member, specify the member name, the password (if required), and the keyword USERID with the new userid. Remember that after the userid for a member has been changed, you may no longer be able to access it.

The CPRS operand turns on the compressed attribute for a member. The operand is needed when a compressed member is added to the VSE/ICCF library file in such a way that VSE/ICCF does not know that the member is compressed. This can happen, for example, when members are punched online from a DTSUTIL archive or backup tape. The member is compressed, but VSE/ICCF is not aware of the fact. In such a case, the compressed attribute must be set on by issuing the /PROTECT command with the CPRS operand. However, any non-compressed records must be deleted from the member (for example, control cards that have been punched into it).

The 'update in progress attribute' (UPIP) for a member may have been left on due to a system failure. To reset this attribute, issue the /PROTECT UPIP command.

Example:

```
*READY
/protect jacdta jaca nopass
*MEMBER CHANGED
*READY
```

/PURGE Command

The /PURGE command removes a member from a library. It is only effective in command mode.

```
/PURge        name [password]
```

name	is the name of the member to be purged (permanently removed) from the VSE/ICCF library file.
password	is the password associated with the member being purged. Must be specified only if the member is password protected.

Password must be specified to purge a password protected member. If a member has been saved as private, only the user who entered the member may purge it.

A public member can be purged by any user who shares the library (unless the alternate security option is in effect).

Common data library members cannot be purged by any terminal user. Library members within the common library can only be purged by the VSE/ICCF administrator.

Example:

```
*READY
/list comment
THIS IS THE MEMBER NAMED COMMENT
*END PRINT
*READY
/purge comment
*PURGED
*READY
/list comment
*FILE NOT IN LIB
*READY
```

RELIST Macro

The RELIST macro routes the contents of the print area, of a print-type member, or of a normal library member to the printer.

```
RELIST     [name [pass] [ICCFSLI]]
```

name	is the name of a library member containing the data to be printed. The member can contain both VSE/POWER JECL and VSE JCL statements. If this operand is omitted, the contents of the print area will be printed. Certain names should be avoided; refer to "Submit-to-Batch Capability" on page 9-29.
pass	is a four-character password which need only be specified if the library member is password protected.
ICCFSLI	applies only to a library member, not to the print area. If ICCFSLI is specified, an * $$ SLI statement will be generated when the VSE/POWER print job is being built. The data will not be written to the VSE/POWER reader queue. Instead, at execution time, VSE/POWER itself will read the data from the VSE/ICCF library file. If ICCFSLI is not specified, the library member will be written into the VSE/POWER reader queue when the print job is being built.

After displaying the output of an interactive partition job you can also print the output. Simply use the RELIST macro to route the contents of the print area to the installation printer. You can also first save the contents of the print area as a print-type library member, and later route it for printing. Specify the member name as the operand of the RELIST macro.

The RELIST macro can also be used to route a normal VSE/ICCF library member to a printer even if it is not print data. This member will always be displayed in 80-character record format (80-80).

The data to be relisted must not contain the at sign (@) in the 1st column.

Notes:

1. *To set the size of the print area, see the /SET buffer system command in Chapter 3. System Commands, Procedures and Macros.*

2. *If the submission of the relist job is not successful, the first two lines of the print area will be overlaid.*

3. *For a print-type member (that is, the member name ends with .P) the first two characters of each 80-character record are interpreted by the DTSRELST utility as control characters and the member will be printed in printline format.*

4. *This macro can only be used in VSE/ICCF systems that support the submit-to-batch capability via VSE/POWER.*

/RENAME Command

The /RENAME command changes the name of a library member.

```
/RENAMe       oldname newname [password]
```

oldname	is the current name of the member.
newname	is the new name that you want to apply to the member.
password	is the 4-character password of the member and need only be specified if the member is password protected. Even though the member name will be changed, the password will remain the same.

This command is only effective in command mode and it can only be used to rename members in libraries to which you have access. It cannot be used to change the names of common members and members of the common library, or of members of a generation member group.

Example:

```
*READY
/input
 ...+....1....+....2
/end
*READY
/save scale publ
*SAVED
*READY
/rename scale s
*RENAMED
*READY
/list s
 ...+....1....+....2
*END PRINT
*READY
```

/RENUM Command (see /RESEQ Command)

[/]REPLACE Command

The [/]REPLACE command replaces a library member with all or part of the input area.

```
[/]REPlace    name [password] [m [n]] [PRIV|PUBL]
```

name	is the name of the library member whose contents are to be replaced by the contents of the input area. The characters '.P' at the end of the member name indicate a print-type member.
password	is a four-character password and need only be specified if the member being replaced is password protected.
m	is a number from 1 to 9999 representing the first line number in the input area to replace the library member.
n	is a number from 1 to 9999 representing the last line number in the input area to replace the library member.
PRIV	indicates that you want the contents of the member to be replaced and it to be given the PRIVATE attribute.
PUBL	indicates that you want the contents of the member to be replaced and it to be given the PUBLIC attribute.

The /REPLACE command (with /) is effective in command or input mode; the REPLACE command (without /) is only valid when you are editing in the input area. If the member is password protected, the 'password' operand must be specified.

To replace the existing library member with only a portion of the input area, specify a beginning and, optionally, an ending line number ('m' and 'n'). If 'n' exceeds the number of lines in the input area, the replacement operation proceeds through the end of the input area. If only one number is specified, it is assumed to be 'm'. In this case, all lines from line number 'm' through the end of the file are affected.

If only a portion of the input area replaces the existing library member, all records not participating in the replace will remain in the input area. The replacing records, however, are deleted from the input area.

The operands PRIV or PUBL can be specified if you want to assign the private or public attribute to the replacement member. If neither PRIV nor PUBL is specified, the previous privacy attribute of the member is retained.

If the '-0' member of a generation member group is being replaced (i.e. the current member), the '-0' member becomes the '-1' member, the '-1' becomes the '-2', and so on.

If the /REPLACE command is used in input mode, input mode is terminated, the replace function is performed and command mode is entered. If REPLACE is used as an editor command (without '/'), the editor is terminated and command mode is entered.

If the library member name specified on the [/]REPLACE command does not exist within the library, an error message is issued. You should then enter the [/]SAVE command.

Example:

```
*READY
/input
09/02/84 840902 84002 SXPT. 02, 1984
/end
*READY
/save daterec publ
*SAVED
*READY
/input
09/02/84 840902 84002 SEPT. 02, 1984
/end
*READY
/replace daterec
*REPLACED
*READY
```

/RESEQ Command

The /RESEQ or /RENUM command resequences, or initially applies, sequence numbers to a member in the library. It is valid in command mode only.

```
/RESeq      name [password] ┌incr ┌col ┌n ┌strt┐┐┐┐
/RENUM                      │100  │73  │8 │    ││││
                            └     └    └  └    ┘┘┘┘
```

name	is the name of the library member to have sequence numbers applied to it.
password	is a 4-character password needed only if the library member ('name') is password protected.
incr	is the resequencing increment. If an increment is not specified, 100 is assumed. The maximum increment is 9999.
col	is the first column where sequence numbers are to be inserted. The maximum is 80-n+1. The default is 73. For members containing alphanumeric data, the maximum is 80-n+1 and the default is 73. For DBCS members, only 1 and 73 are valid starting columns.
n	is the number of columns for the sequence number field. The maximum is 16 and the default is 8. If you resequence a DBCS member and if col has a value of 73, n must be 8.
strt	is the beginning sequence number. If this operand is omitted the increment becomes the starting sequence number.

If the last four operands are not specified, sequence numbers are placed in columns 73-80. The increment value will be 100 and the initial sequence number will be equal to the increment. If the 'n' operand is specified, the 'incr' and 'col' operands must also be specified. Likewise, if 'strt' is specified, then the 'incr', 'col' and 'n' operands must also be specified.

Do not use this command for resequencing VS BASIC programs, because sequence numbers within the instructions themselves will not be altered to reflect the new sequence numbers. See the description of the VSBRESEQ procedure on page 3-148.

Notes:

1. *The /RESEQ command inserts new sequence numbers into each of the 80-byte records of the member. It makes no difference whether the member is print-type or not. That is, print data is treated the same way as 80-character records.*

2. *Statements with a '/' in column 1 will not be resequenced if the sequence number starts below column 73.*

Examples:

1. Resequence a password protected member in the standard sequence location:

```
*READY
/reseq utilprg jaca
*RESEQUENCED
*READY
```

2. Resequence a COBOL program in the standard COBOL sequence location (colums 1-6):

```
*READY
/res cobprog 100 1 6
*RESEQUENCED
*READY
```

/RETRIEV Command

The /RETRIEV command places a previously entered command in the terminal input area. There, the retrieved command may be modified and then reissued by pressing ENTER.

```
/RETriev    [OFF]
```

OFF terminates the retrieve function and clears the internal stack of previously entered commands.

The /RETRIEV command is valid in the following modes:

```
-   CM (command)
-   LS (list)
-   SP (spool)
-   EX (execution)
-   RD (conversational read)
```

It is recommended to define a PF key for the /RETRIEV command.

Before any command can be retrieved, the retrieve function must be set on once after logon (and also after you issued /RETRIEV OFF). You do this by entering /RETRIEV without an operand. Note that with this first /RETRIEV you do not yet retrieve a command.

When the retrieve function is on, the /RETRIEV command retrieves the command that had been entered last. Issuing /RETRIEV again (without entering any other command) retrieves the command that had been entered second to the last, and so on.

The number of commands that can be accumulated depends on the size of the commands. For example, at an average command size of 20 characters, 12 commands can be accumulated. The maximum size for a command to be retrieved is 255 characters. If the internal stack is full, each new command replaces one or more of the oldest commands in the stack.

Only commands entered via the ENTER key can be retrieved. Multiline input is split up into single commands. Therefore, each command of multiline input must be retrieved separately.

Example:

```
/set pf12 /ret        (press ENTER, define PF key as /RETRIEV)
                      (press PF key 12, sets retrieve function on)
/lib                  (press ENTER)
submit ...            (press ENTER)
/list ...             (press ENTER)
                      (press PF key 12)
/LIST                 (retrieved at terminal)
                      (press PF key 12)
SUBMIT ...            (retrieved at terminal, press ENTER)
                      (press PF key 12)
SUBMIT ...            (retrieved at terminal)
                      (press PF key 12)
/LIST ...             (retrieved at terminal)
```

/RETURN Command

The /RETURN command is only valid if you entered VSE/ICCF through the interactive interface of VSE/System Package (VSE/SP). The command brings you back into that interface.

```
/RETURN
```

The /RETURN command is effective in command mode.

Typically, you would leave the interactive interface of VSE/SP if you wanted to enter a VSE/ICCF command, macro or procedure that is not supported in that interface. The /RETURN command facilitates a fast return to VSE/SP.

If you changed the environment as used by the interactive interface of VSE/SP (for example, you changed the PF key settings), you should return to that interface via the /LOGOFF command. You would thus maintain your system's integrity.

/ROUTEP Command

The /ROUTEP command routes a VSE/POWER queue element to either the installation printer or punch, or to a remote VSE/POWER RJE work station. It is valid in command mode only.

```
/ROUTEP   queue jobname [jobnumber [jobsuffix]]
/RP                     [CLASS=jobclass] [REMOTE=remid] [PWD=password]
```

queue	indicates the VSE/POWER queue from which the job is to be routed:
	`LST for list queue`
	`PUN for punch queue`

jobname	is the 2 to 8 character name of the VSE/POWER job to be routed.
jobnumber	specifies the 1 to 5 digit job number assigned to the job by VSE/POWER. Must be specified if jobname is not unique within queue.
jobsuffix	is the 1 to 3 digit job suffix which designates the segment number. If this parameter is omitted, the entire job will be routed. A suffix should not be specified if the queue entry is not segmented.
CLASS=jobclass	is a letter from A to Z indicating the VSE/POWER target output class. Default is class A[10].
REMOTE=remid	is a work station number from 1 to 250 indicating a remote VSE/POWER RJE work station identification. If this parameter is omitted, REMOTE=0 is assumed which stands for the system printer.
PWD=password	specifies the 1 to 8 alphanumeric password assigned to the VSE/POWER job when it was submitted.

/RP routes output from the VSE/POWER output queues to the specified output class and to a specified work station, and the disposition is changed to DELETE. If a local or remote writer task is active for the target output class, the output will be printed or punched and removed from the | VSE/POWER queue. Mixed data should only be routed to a 3200 printer after having been prepared | for printing, for example, with Utility PRPQ (5799-BGF).

Notes:

1. *The displayed output associated with this command can also include the VSE/POWER message. These messages are identified by the characters '1Q', '1R' and '1V' followed by a message code. These responses are explained in "VSE/POWER Remote Job Entry User's Guide".*

2. *Certain job names should be avoided; refer to "Submit-to-Batch Capability" on page 9-29.*

Example:

Route the print output of 'MYJOB' from class Q[10] to class A[10] to get it printed.

```
/routep lst myjob
*OK
*READY
```

10 For your installation a different default may have been defined.

/RP Command (See /ROUTEP Command)

RPGIAUTO Procedure

The RPGIAUTO procedure causes a RPG II source program to be preprocessed by the AUTO REPORT feature and then to be compiled by the RPG II compiler.

```
RPGIAUTO     name1 [OBJ name2|OBJ *] [options]
```

name1	is the name of a member in the library containing the RPG II source program to be compiled.
name2	is the library member name under which the object module resulting from the compilation is to be placed. This member need not initially exist. If the named member already exists, the object module will overlay it.
OBJ *	indicates that the object module is to be placed in your punch area. The punch area is also the default if this parameter is omitted.
options	are one or more /OPTION job entry statement options that alter the way the compiler is processed (e.g., NODECK, LIST, etc.).

The 'options' operand can be specified as one or more options which influence the compilation, such as NODECK, LIST, RESET, etc. If the NODECK option is specified, no object module will be produced even if the OBJ operand is present. The NORESET option should be specified if you are stacking multiple object decks in the punch area.

The LOAD procedure can be used to load and run the object module produced by this procedure.

Examples:

1. Compile the RPG II source program contained in the library member 'AUTOPGM' and place the resulting object module in the punch area:

   ```
   rpgiauto autopgm
   ```

2. Compile the RPG II source program contained in library member 'PROGR', place the resulting object module in library member 'OBJMEMB' and list the input source program on SYSLST:

   ```
   rpgiauto progr obj objmemb list
   ```

RPG II Procedure

The RPG II procedure causes a library member to be processed by the DOS/VS RPG II compiler.

```
RPGII          name1 [OBJ name2|OBJ *]  [options]
```

name1	is the name of a member in the library containing the RPG II language source program to be compiled.
OBJ name2	is to be the library member name of the object module resulting from the compilation. This member need not initially exist.
options	are one or more /OPTION job entry statement options that alter the way the compiler is processed (e.g., NODECK, LIST, etc.).

The object module will be placed in your punch area if the OBJ operand is omitted or if OBJ * is specified. If the OBJ operand specifies a member name ('name2' parameter), the object module will be saved in the library under this name. If the named member already exists, the object module will overlay it.

The 'options' operand can be specified as one or more options which influence the compilation, such as XREF, LIST, SYM, etc. If the NODECK option is specified, no object module will be produced even if the OBJ operand is present. The NORESET option should be specified if you are stacking multiple object decks in the punch area.

The LOAD procedure can be used to load and run the object module produced by this procedure.

Examples:

```
1.  rpgii rpgtest

2.  rpgii xxprog obj objmem list
```

RPGIXLTR Procedure

The RPGIXLTR procedure prepares RPG II source program that will call the DL/I Translator for processing by the RPG II compiler.

```
RPGIXLTR     name1 [PUNCH name2|PUNCH *]
```

name1	is the name of a member in the library containing the RPG II source program to be translated.
PUNCH name2	is the library member name into which the translated source module is to be placed. This member need not initially exist. If the named member already exists, the source module will overlay it.
PUNCH *	indicates that the source module resulting from the DL/I translation is to be placed in your PUNCH area. The PUNCH area is the default if the PUNCH parameter is omitted.

If your source program calls the DL/I Translator, it must first be processed by the translation procedure before it can be processed by the RPG II Compiler (or by the RPGIAUTO procedure).

Examples:

1. Translate the RPG II source program contained in library member 'RPGDLI' and place the translated source module in the punch area:

   ```
   rpgixltr rpgdli
   ```

2. Translate the RPG II source program contained in library member 'PROGR' and place the translated source module in the library under the name 'DECK':

   ```
   rpgixltr progr punch deck
   ```

3. Translate the RPG II source program contained in the library member 'PROGR' and place the translated source module in your PUNCH area:

   ```
   rpgixltr progr punch *
   ```

RSEF Procedure

The RSEF procedure calls the RSEF program (the RPG II Source Entry Facility).

```
RSEF            [nn]
```

nn is the GETVIS space in multiples of 1K bytes reserved for execution of the RSEF program. The default is 60K bytes.

After you have entered RSEF you will be prompted for filename and password.

The *DOS/VS RPG II User's Guide*, SC33-6074 describes how to use the RPG II Source Entry Facility.

/RUN Command

The /RUN command causes the specified phase to be loaded from a VSE library and run. With no operand, it causes the job or job stream in the input area to be run. It is only effective in command mode.

```
/RUN          [phasename [data]]
$
```

phasename	is the 1 to 8 character name of a phase to be loaded from a VSE library and run.
data	is from 1 to 72 columns of data to be passed to the phase being loaded as the first and only 80-character input record.

If the 'data' operand is specified, 'data' will be passed to the phase being loaded and run as the first and only 80-character input record read from the job stream.

If the '$' abbreviated form of the command is used, no space is needed between the '$' and the phasename. Thus, $XYZ will cause the program named XYZ to be run.

The /RUN or $ command will cause a job stream to be built in the input area and run. Thus, any prior contents of the input area will be destroyed. For example, the command:

```
/RUN DTSUTIL DISPLAY USERS
   or
$DTSUTIL DISPLAY USERS
```

is functionally equivalent to the following set of VSE/ICCF commands:

```
/INPUT
/LOAD DTSUTIL
 DISPLAY USERS
/ENDRUN
```

Note that column 1 of the data operand begins one space after the phasename.

Once the /LOAD and 80-character input record have been placed within the input area, the job can be re-run simply by entering /RUN or $ with no operands; in this form the command will no longer destroy the contents of the input area.

Examples:

```
*READY
$SSERV   DSPLY A.SORSBK
*RUN REQUEST SCHEDULED
  .
  . (output from SSERV program)
  .
*READY
/input
/load vfortran
;write (3,10) (note that ; is the logical tab character)
10;format (' this means the program ran')
;end
/end
*READY
/run
*RUN REQUEST SCHEDULED
  (compiler output)
TOTAL MEMORY REQUIREMENTS 000154 BYTES
HIGHEST SEVERITY LEVEL OF ERRORS FOR THIS MODULE WAS 0
****** BEGIN LINKNGO
  12,235 BYTES REQUIRED FOR PROGRAM STORAGE
  PGM LOADED AT 09B100
  XFER ADDRESS 09B100
******
THIS MEANS THE PROGRAM RAN
**** JOB TERMINATED - RETURN CODE 00 NORMAL EOJ
*READY
/run   (issuing /RUN again will cause the job to be rerun)
```

[/]SAVE Command

The [/]SAVE command saves all or part of the contents of the input area as a member in the library.

```
[/]SAve        name [password] [m [n]] [PRIV|PUBL]
```

name	is the 1 to 8 character name to be applied to the member once it is saved in the library. It must begin with an alphabetic character. The characters '.P' at the end of the member name indicate a print-type member.
password	is a 4-character password that you can specify if you want the library member to be password protected. Note that the password cannot be PRIV or PUBL.
m	is a number from 1 to 9999 which represents the first line number in the input area to be saved. If 'm' is not specified, the first line saved is line 1. If only one number is specified, it is assumed to be 'm'.
n	is a number from 1 to 9999 which represents the last line number in the input area to be saved. If 'n' is not specified, or if the number specified exceeds the number of lines in the input area, the last line saved is the last line in the input area.
PRIV	indicates that the member is to be PRIVATE.
PUBL	indicates that the member is to be PUBLIC.

The /SAVE command (with /) is effective in command or input modes. Without the slash (/), SAVE is valid only if you are editing in the input area. The command has no meaning for a library member, since the member is already in the library and commands processed against it have already taken effect.

All lines not saved remain in the input area, and all lines saved are no longer in the input area at the end of the SAVE operation.

The operands PRIV or PUBL can be specified if you want to save the data as private or public. If neither PRIV nor PUBL is specified, the data is saved according to the specification in your user profile.

If the /SAVE command (with /) is used in input mode, input mode is terminated, the data is saved and command mode is entered. When the SAVE command (without /) is entered, the editor is terminated and command mode is entered.

If the member name specified in the [/]SAVE command is the '-0' (i.e., current) member of a generation member group, the '-0' member becomes the '-1' member, the '-1' becomes the '-2', etc. The oldest member of the group will be purged. The [/]SAVE and [/]REPLACE commands have the same effect for generation member groups.

If the name specified in the [/]SAVE command already exists within the library, an error message will be issued and the input area will not be saved. To replace a library member with the contents of the input area, use the [/]REPLACE command.

If your library or directory is full when the [/]SAVE is entered, an error message is issued. Use the /PURGE command to remove unnecessary library members thus freeing library and directory space. Then reenter the [/]SAVE command.

Examples:

```
*READY
/input
/load vfortran
      write (3,10)
10    format (' this means the program worked')
      end
/save testprog
*SAVED
*READY
/exec testprog
*RUN REQUEST SCHEDULED
   .
   .  (compiler and LINKNGO output)
   .
THIS MEANS THE PROGRAM WORKED
**** JOB TERMINATED - RETURN CODE 00 NORMAL EOJ
*READY
/input
/insert $$PUNCH   (place the object deck from the above
                  compile in the input area)
*END INSERT
/end
*READY
/save testobj
*SAVED
*READY
/exec testobj    (rerun the program from the object deck)
*RUN REQUEST SCHEDULED
  (LINKNGO output)
THIS MEANS THE PROGRAM WORKED
**** JOB TERMINATED - RETURN CODE 00 NORMAL EOJ
*READY
```

SCRATCH Procedure

The SCRATCH procedure removes DISP=KEEP files (specified in the /FILE statement) from the dynamic space area after they are no longer required.

```
SCRATCH      fileid|* volser [strt nspace [PURGE]]
```

fileid	is the file identification as it appears in the disk VTOC of the file to be scratched. '*' should be specified when using the PURGE option.
volser	is the volume serial number of the volume on which the file is located.
strt	is the starting track/block number of the file and need only be specified if DASD file protection is active (DASDFP=YES in the IPL command SYS) or if PURGE is specified.
nspace	is the number of tracks (for CKD) or the number of blocks (for FBA) in the file and need only be specified if 'strt' is specified.

SCRATCH can be used by the VSE/ICCF administrator to scratch areas of the dynamic space area even when no files are associated with the area (PURGE option). This may be necessary if a permanent area was allocated but the job was canceled before the file was actually opened.

To scratch a file, enter the file identification (ID= parameter when the file was created) followed by the serial number of the volume on which the file resides. If DASD file protection is active in your system, the serial number must be followed by the starting track/block and the number of tracks/blocks in the file.

To purge an area of dynamic space - whether or not an actual file exists in the area - specify the file name as '*'. The volume serial number and location of the area must be specified with the PURGE operand.

Examples:

```
1.  scratch myfile etsvol             (scratch a file)

2.  scratch payfile 111111 36 12      (scratch a file if protect)

3.  scratch * 111111 240 36 purge     (scratch an area)

4.  scratch filename.userid.termid    (if no 'fileident' was specified
                                      in the /FILE statement at file
                                      creation time)
```

SDSERV Procedure

The SDSERV procedure displays sorted directory information from your primary or connected library, or from the system common library.

```
SDSERV       ┌    ┐  ┌    ┐
             │*abc│  │NAME│
             │CONN│  │USER│
             │COM │  │DATE│
             └    ┘  └    ┘
```

*abc	displays directory information for members starting with 'abc'. 'abc' can be a string from 1 to 7 characters.
CONN	directory information from your connected library is displayed.
COM	directory information from the common library is displayed.
NAME	orders the display by member name.
USER	orders the display by userid.
DATE	orders the display by date of entry or change.

If the second operand is omitted, the order will be by member name. If no operands are specified, your primary library directory will be sorted by member name and displayed.

Examples:

```
1. sdserv            (display of primary library directory
                      sorted by name)
2. sdserv conn date  (display of connected library sorted by date)
3. sdserv date       (display of primary library sorted by date)
4. sdserv *r710      (display of primary library member names
                      beginning with R710)
```

/SEND Command

The /SEND command sends a message to the system operator or to another terminal user.

```
/SENd          [userid|COPER|ALL]    message text
```

userid	specifies the terminal user who is to receive the message. If the user is not logged on, the message will be saved and displayed automatically when the user next logs on. If the user is logged on, the message will be displayed automatically, provided automatic message display is in effect. If non-automatic message display is in effect, the message will be saved and displayed when the user next logs on. Or the user can display the message with the [/]MSG command.
COPER	specifies that the message is to be sent to the system operator.
ALL	specifies that the message is to be sent to every logged-on terminal user (VSE/ICCF administrator only).
message text	is the message to be sent. The maximum length is 246 characters if userid is specified, 247 if the 'ALL' operand is used, or 80 if the message is to be sent to the system operator. The message text can only contain alphanumeric characters.

The /SEND command is valid in command, list, and execution modes.

Messages should only be sent to the system operator when absolutely necessary, since they could 'lock up' VSE/ICCF if they are not handled promptly.

The /SEND command is not effective if used within a procedure.

Example:

```
*READY
/send coper this is a message to the system operator
*MESSAGE SENT
*READY
```

[/]SET Command

The [/]SET command has five forms which can be used to set control characters, VSE/ICCF features, the editor autoinsert features, system control features, and 3270 screen features. The [/]SET command is valid in all modes except in message mode.

SET CONTROL CHARACTERS

```
[/]SET        BS=char|DEL=char|END=char|ESC=char|HEX=char|TAB=char
[/]CTL
```

SET VSE/ICCF FEATURES ON OR OFF

```
[/]SET        BYPass|DATanl|EXTab|COMlib|IMPex|VERify|MSGauto [ON|OFF]
[/]CTL
```

SET EDITOR AUTOINSERT FEATURE

```
SET           AUtoinsrt [ON|OFF]
CTL
```

SET SYSTEM CONTROL FEATURES

```
[/]SET        BUFfer [bufsiz|REset|OFF]
[/]CTL        CAse OFF|REset

              CAse INput [UPper|REset|MIxed]

              CAse [INput] UPper|REset|MIxed

              CAse OUTput [UPper|MIxed|REset
                          [NOctl|REset|CTl]]

              CLAss {class[WAIT|NOWait]|REset|OFF|WAIT|NOWait}
              DELAY [TIme|time|STop|BYpass|REset|OFF]
              LINesize lnsize

              LOG ┌   ┐┌                   ┐┐
                  │OFF││                   ││
                  │ON ││INPut|OUTput|INOut││
                  │nnn││                   ││
                  └   ┘└                   ┘┘

              PF|PFLS|PFED|PFEX  CLEar|ON|OFF|SAVE|RESTORE
              PFnn|PFnnLS|PFnnED|PFnnEX  function|CLEar|OFF
```

```
[/]SET      SCReen  ┌                              ┐
[/]CTL              │REset|OFF                     │
                    │ERase|NOERase                 │
                    │ALarm|NOALarm                 │
                    │CLear                         │
                    │i [d1 [d2 [c1 [c2]]]]         │
                    │COLumn c1[c2]|REset           │
                    │ROw    r1[r2]|REset           │
                    └                              ┘
```

char	is a single character which must not be alphabetic, numeric or already set as a control character. Also characters with special significance within VSE/ICCF cannot be used: `$ @ * / ( ) = ,(comma) blank '(quote)` The BS (backspace) character should not be set to '&'. If you use print-type members you should use no '.', because this is part of the print-type member name. If 'char' is specified as OFF, the effect of a previous [/]SET is removed.
ON	sets the feature on.
nnn	is a value between 20 and 300 which sets the number of lines in the log area before logging wrap around occurs. If not specified, or if ON is specified, the default size is 300. It can be specified only once and only before logging is set on for a session.
OFF	sets the feature off.
REset	sets the feature to the various defaults (described for each parameter).
bufsiz	is a decimal number indicating the size of your print area (i.e., number of print lines).
class	is an alphabetic character from A to Z indicating the interactive partition scheduling class to be associated with your terminal.
time	indicates the time delay between the displays of 2 successive screens and is measured in seconds and expressed as a decimal number from 1 to 255.
lnsize	is a decimal number from 1 to 80 indicating the line width for displays associated with the /DISPLAY and /LIST commands and the input mode VERIFY feature. If this value is greater than 72 it indicates the default for PRINT and VERIFY editor commands.
function	is any command or line of data to be associated with a given PF key. The maximum length of the command or data string is 80 characters.
i	is a decimal number from 1 to 4 indicating the number of lines on the screen to be used as an input (or editor command) area.
d1	is a decimal number from 3 to 39 indicating the starting physical line number of the output (or display) area of the screen.

d2	is a decimal number from 5 to 41 indicating the number of lines in the output (or display) area of the screen.
c1	is a decimal number from 1 to 132 indicating the first column of the output (or display) area on the screen.
c2	is a decimal number from 1 to 132 indicating the last column of the output (or display) area on the screen.
r1	is a decimal number from 1 to 37 indicating the first row of the output (or display) area of the screen. Row '1' refers to the first row below the scale line, not the first row of the screen.
r2	is a decimal number from 5 to 41 indicating the last row of the output (or display) area on the screen. Row '1' refers to the first row below the scale line, not the first row of the screen.

The [/]SHOW command can be used to display the current settings of most [/]SET command functions.

The /SET command has no effect if issued from a program running in an interactive partition, or from a procedure.

Set Control Characters

This form of the [/]SET command is used to set any of the control characters that affect the entry of data or commands from the terminal.

BS=char is used to alter the logical backspace character. For hardcopy terminals (2740, 2741, 3767), the backspace character is usually set to the Backspace key. However, you can set any other character as the backspace character. The backspace character is not usually required for display terminals.

Example:

```
*READY
/set bs=#
*CONTROL SET
*READY
/input
abc##de#f
/list
ADF
```

DEL=char defines a line delete character which, when used as the last character of an input line, causes the input line to be ignored. The line delete character feature is more useful on hardcopy type terminals (2740, 2741, 3767) than on the 3270 terminal.

The END=char defines a logical line end character, which allows you to type in several commands and/or lines of data without having to press the ENTER or EOB key after each line. It causes the lines to be sent to the system and processed one by one until the last logical line has been processed. (If you are using a 3270, see also the /SET DELAY command, which sets a system control feature).

Example:

```
*READY
/set del="
*CONTROL SET
*READY
/set end=#
*CONTROL SET
*READY
/input#aaa#bbb#ccc#/end#/list
*READY
AAA
BBB
CCC
*END PRINT
*READY
```

ESC=char defines an escape character, which indicates that the following character is not to be treated as a control character even though it is set to a control character function. See example for HEX operand.

HEX=char defines a hexadecimal entry character which, when it occurs in terminal input, causes the two following characters to be treated as hexadecimal digits forming a single EBCDIC character. If either of the characters following the 'hex' control character is not a hexadecimal digit, the 'hex' character will be treated as an ordinary data character. Note that, to use unprintable characters in the member name (for example for security reasons), the characters following the hex character must be greater than hex '40'.

Example:

```
*READY
/set esc=#
*CONTROL SET
*READY
/set hex=?
*CONTROL SET
*READY
/input
xyz#?abc?45?2c
/end
*READY
/listx
XYZ?ABC..
EEE6CCC42
789F1235C
*END PRINT
```

TAB=char allows you to set the logical tab character. When the logical tabs are set using the /TABSET command, use of the logical tab character in an input line will cause this line to be separated into fields according to where the tab positions are set. If no logical tab character is set, the Tab key on the terminal (Field Mark key on the 3270) is the default for logical tabbing.

Example:

```
*READY
/input
/tabset  10  16  36
/set tab=;
main;start;0
;balr;5,0
;using;*,5
;l;1,0(1)
;eoj
;end
/end
*READY
/list
MAIN    START 0
        BALR  5,0
        USING *,5
        L     1,0(1)
        EOJ
        END
*END PRINT
*READY
```

Set VSE/ICCF Features

This form of the [/]SET command is used to set various VSE/ICCF features ON or OFF. If the command is specified without either ON or OFF, ON is assumed.

BYPASS feature - When this feature is ON it causes all VSE/ICCF control character interpretation on input to be bypassed. In other words, if tab, backspace or other control characters occur in an input line, the control character function will not be performed. Instead, the control characters will be treated as ordinary data. This feature is initially set off, that is, it must be explicitly set on.

COMLIB feature is normally ON. When you set this feature OFF, directory look-ups will not include the common library. Only your primary and connected library will be searched.

DATANL feature only applies to 3270 terminals with the data analysis or text features. It is normally OFF but you should set it ON before starting any input or output operations including special APL or TEXT mode characters. ON causes VSE/ICCF terminal input and output routines to scan input and output lines for the special 3270 data analysis character set. Otherwise, the APL alternate characters enter VSE/ICCF as some character preceded by a 1D control character. If this feature is set ON, the characters preceded by a 1D will be translated to unused locations within the EBCDIC character set. On output, if the feature is set ON, these translated characters will be reconverted to a 1D followed by the actual terminal character.

EXTAB (execution tabbing) feature allows you to set execution mode tabbing on. Normally, when a job is executing in an interactive partition and it issues a conversational read, the input data will not be scanned for tab characters. Setting this feature ON forces tabbing to be enabled for conversational reads. This feature will automatically be set OFF at the end of each interactive partition execution.

IMPEX (implied execute) feature is normally set ON. It allows a procedural CLIST to be invoked as if it were a command. In other words, when this feature is set ON (as is normally the case), the command '/EXEC PROCA CLIST PARMA PARMB' can be entered as PROCA PARMA PARMB . Setting this feature OFF prevents an automatic library search when an input command appears to be invalid.

MSGAUTO feature lets you control the display of messages received for your userid. When set ON, messages are displayed automatically before any other terminal operation is performed. Message mode (MS) is entered, the screen is saved and messages are displayed until you acknowledge the receipt by pressing ENTER, then the screen is restored and normal processing is resumed. This setting should be used sparingly, however, since it is an extra burden on the traffic on the line, especially for remote terminals.

When set OFF, it indicates that messages are not to be displayed when they are received, but are to be kept. Such messages can be displayed with the [/]MSG command.

The default is the option specified in your user profile. Common messages issued by the system operator are always displayed automatically. Messages are deleted after they have been displayed and cannot be looked at again, unless they were routed to a 328x hardcopy printer. This is possible if they are displayed via the /MSG command.

Regardless of the MSGAUTO feature setting, messages are always displayed automatically when you log on to VSE/ICCF.

VERIFY feature - causes the last 10 lines of input to be displayed on the 3270 terminals each time a new input line is entered. The VERIFY feature is automatically set ON for local 3270 terminals. For non-3270 terminals, setting the VERIFY feature ON causes the last input line to be repeated back to your terminal. This can be useful for verifying that tabbing or other control character functions are working properly. The VERIFY feature is normally OFF for remote 3270s and for non-3270 terminals.

Examples:

```
/set impex OFF
*CONTROL SET
*READY
/set verify
*CONTROL SET
*READY
/set tab=;
*CONTROL SET
*READY
/tabset 5 10 15
*TABS SET
*READY
/input
a;b;c
A    B    C   (because VERIFY is on)
```

Set Editor Autoinsert Feature

AUTOINSRT feature when set ON causes any data processed by the editor that is not an editor command to be processed as if it were part of an INSERT command. That is, it will be inserted into the file just as if an INSERT command had preceded it. This feature must be set on for each editing session; it is normally off. If the command is specified with no operand, ON is assumed.

Set System Control Features

This form of the [/]SET command is used to set the values of various system control features.

BUFFER feature lets you control the size of your print area. The 'bufsiz' operand can be specified as small as 20 lines for hardcopy terminals and 60 lines for 3270 terminals. The upper limit depends on how VSE/ICCF was installed, or on the value in your user profile. Use the /USER PROFILE command to check this limit. If your print area is too small, print output from interactive partition jobs will be displayed more often. This can mean that print output that you want to save will be incomplete. A larger print area allows you to store all of a printed report for display. This report can then be saved as a member of your library. You can also transfer it to VSE/POWER for printer output (RELIST macro), or to a local hardcopy terminal (/HARDCPY command). To reset your print area to the default size, specify RESET. The default may have been set as a VSE/ICCF tailoring option. This default results if you do not specify the bufsize parameter.

CASE feature allows you to vary the character translation performed during terminal input or output (the CASE operand is ignored in full screen edit mode). The defaults are set so that lower case input data is translated to upper case. The default for output data is set such that no upper case output translation occurs; however, terminal control characters (such as backspace or carriage return) will be removed from the output. These defaults can be altered as follows:

- Specify /SET CASE INPUT MIXED to allow lower case input data.
- Specify /SET CASE INPUT UPPER or RESET to set input translation back to the default upper case mode. If the DBCS attribute is set for a member, this command will be ignored and the member will be treated as if case were set to mixed.
- Specify /SET CASE OUTPUT UPPER to cause output data which can be in lower case internally to be displayed as upper case. If the DBCS attribute is set for a member, this command will be ignored and the member will be treated as if case were set to mixed.
- Specify /SET CASE OUTPUT MIXED or RESET to set output translation back to the initial default.
- Specify /SET CASE OUTPUT MIXED or UPPER CTL to cause control characters imbedded in the data (new line, backspace, carriage return, tab, line feed) to appear untranslated, thus allowing you some control over hardcopy terminal output formatting. But do not attempt upper case translation for output.
- Specify /SET CASE OUTPUT CTL to cause control character and output character translation.
- Specify /SET CASE OUTPUT MIXED or UPPER NOCTL or RESET to return to normal output control character translation.
- Specify /SET CASE RESET or OFF to return both input and output translation to the defaults described above.
- The SET CASE editor command sets overall system character translation. To vary character translation for the present session only, use the CASE and IMAGE editor commands.

CLASS feature sets your interactive partition scheduling class. Each interactive partition is associated with one or more classes. When you run a job (with the /RUN or /EXEC command), it is only executed in an interactive partition associated with a class that matches your class. By varying the CLASS parameter with the /SET CLASS command, you can direct your job to interactive partitions with characteristics that you need. For example, you may need a larger partition or one with preallocated work files. Your VSE/ICCF administrator will tell you the interactive partition classes to which you have access.

When you request to run a program in an interactive partition, an interactive partition which matches your class may currently not be available. Through /SET CLASS, you can indicate ahead of time whether you want to wait or continue in command mode. The WAIT operand tells VSE/ICCF that you want to wait - in execution mode - until an interactive partition becomes available (WAIT is the default). The NOWAIT operand indicates that you want to take back the run request and do other work in command mode. You can either name a specific CLASS 'class', or omit the 'class' specification in which case the currently set class is addressed.

The RESET or OFF operands return you to the default class as specified in your user profile and set this class to 'WAIT'.

DELAY feature lets you delay the output to the terminal of commands or lines of input. Normally, all commands or lines in a macro, or in multiple line input, are processed one after the other -- that is, with no pause between. To create a pause, use the following operands:

- DELAY TIME delays the output to the terminal from each command for three seconds, which lets you see the output before it is overlayed. To vary this interval, specify a value between 0 and 255 instead of the TIME operand.

- DELAY STOP halts processing between each line. Press ENTER to display the results of the next command in the macro or input sequence.

- DELAY BYPASS allows the output from only the last command or line to be transmitted to your terminal from either multiple line input, or commands from a macro.

- DELAY RESET or OFF turns off the DELAY setting so that no delay occurs. If no operand is specified, RESET is assumed.

The SET DELAY command has no effect in full screen edit mode. In macro processing, output of intermediate results is suppressed by a @NOPRINT macro order.

LINESIZE feature varies the line width used for list mode display functions, including the /DISPLAY and /LIST commands and the input mode VERIFY function. If columns 73-80 are blank or numeric, the default linesize is 72, otherwise it is 80 (or whatever is specified in the user profile). The value specified must be between 1 and 80.

The LOG feature lets you control terminal logging, which is very useful for 3270 terminal users, who have no way of obtaining a hardcopy record of their terminal activities. It can be specified only once and only before logging is set on for a session.

- LOG ON sets the basic logging (ON can be omitted if no operands follow). All VSE/ICCF commands that you entered or which you issued from a macro are logged.

- LOG ON INPUT will cause input data you enter in input or edit mode to be logged along with the basic logging. To log only command responses, specify LOG ON OUTPUT. If you want to log both input and output, specify LOG ON INOUT.

- LOG OFF sets all forms of logging off, which is the usual setting.

- The log area contains an identifier (I or O) indicating whether the line was input or output; the time when the logging occurred; the mode of your terminal during logging, and the first 64 characters of the command, data or output line. To view your log area, issue the /LIST $$LOG command.

PF [ED|EX|LS] feature allows you to manipulate the PF key settings for command mode (PF), for edit mode (PFED), for execution and conversational read mode (PFEX) and for list and spool mode (PFLS). If CLEAR is specified, the named PF key settings are cleared. If OFF is specified, the named PF key settings are suspended, ON will reinstate them again. If SAVE is specified, the specified set of PF-keys is saved in the PF-key save area, the currently valid PF-key set remains unchanged, and the previously saved values will be overwritten. If RESTORE is specified, the specified set of PF-keys is replaced by the content of the PF-key save area and the specified set of PF-keys with the restored values is activated; the content of the PF-key save area remains unchanged. But keep in mind that the use of the VSE/ICCF HELP macro changes the contents of the PF-key save area. If PF key settings are cleared or suspended, the next in the hierarchy become available (see Figure 3-3).

MODE	Corresponding PF Key Setting Hierarchy
CM	PF
ED,FS	PFED --> PF
EX,RD	PFEX --> PF
LS,SP	PFLS --> PF --> default setting (see Figure 2-1)

Figure 3-3. PF Key Setting Hierarchies

PFnn[LS|ED|EX] feature allows you to set a PF key to a function or to clear this setting. The 'nn' must be a decimal number from 1 to 24. [LS|ED|EX] defines the mode to which it belongs. The function can be any single command or line of data. Setting a PF key in this way and pressing it has the same effect as entering the command or data line to which it is set. CLEAR or OFF terminates the previous setting, after which the next lower PF key setting in the hierarchy becomes active, provided it is defined and is not set off.

- Only in full screen editor mode is it possible to set PF functions to handle multiple lines separated with end-of-line characters. You can, however, set a program function to a macro that consists of more than one line or command.

- The [/]SET PFnn command translates all characters specified in the 'function' field to upper case. In edit mode you can enter both upper and lower case characters using the SETLC PFnn command. You must specify the 'PF' as upper case, and the CASE feature must be set to MIXED. (The SETLC command is not described any further in this manual.)

- If your terminal does not have PF keys, you can still set PF functions and invoke them via the [/]PFnn command. These functions are also valid in edit mode.

- If you use a double ampersand ('&&') when setting a PF key, data entered before the key is pressed will replace the '&&'. This allows you to set more general program functions. If you specify data when setting a PF key and no '&&' appears within the PF key setting, the data will be added to the end of the PF key setting before the command or data line is processed. If, however, a valid command is entered in the command area in full screen edit mode it will not replace the '&&'. It will be processed as an editor command. Similarly, if you set on the AUTOINSRT feature, data of the command area are inserted after the current line pointer and again will not replace '&&'.

Notes:

1. *When setting CASE, remember that not all 3270s can display lower case characters. Also your installation may have set up your terminal to accept only upper case data.*

2. *PF keys cannot be set and accessed within a procedure. They can, however, be set and accessed within a macro.*

Examples:

```
*READY
/set case input mixed
*CONTROL SET
*READY
/input
aaa
bbb
/end
*READY
/list
aaa
bbb
*END PRINT
*READY
/set case output upper
*CONTROL SET
*READY
/list
AAA
BBB
*END PRINT
*READY
/set pf1 /list 1 5 $$log
*CONTROL SET
*READY
/pf1  (or pressing the PF1 key)
I 09/19 CM  /list 1 5 $$LOG
I 09/18 CM  /list
I 09/18 CM  /list
I 09/17 IN  /end
I 09/17 CM  /input
```

Set 3270 Screen Features

This form of the [/]SET command is used either to set the screen input or output areas, or to set screen oriented features ON or OFF. The /SET SCREEN command applies only to 3270 terminals. The full screen editor formats the screen independently.

With the /SET SCREEN command you can divide your screen into separate areas, which allows you, for example, to display information in one part of the screen, return to command mode, and then transfer the active part of the screen to another screen area.

The maximum values that can be set for output areas depend on the model of 3270 that you are using. The values for the various models of the 3270 follow:

```
     Model                          r1    r2        d1    d2
  3278 Model 5                     1-21  5-25      3-23  5-25
  3278 Model 4                     1-37  5-41      3-39  5-41
  3278 Model 3                     1-26  5-30      3-28  5-30
  All others                       1-18  5-22      3-20  5-22
```

The setting of the screen parameters in the context editor will have a lasting effect after edit mode is terminated.

The specification of these parameters will override any screen width specifications in the editor VERIFY and PRINT commands.

Note: For list-type commands (such as /LIBRARY, /LIST), the last two rows of a screen are reserved for messages, and hence are not available for display purposes.

VSE/ICCF uses the wide screen of the IBM 3278 Model 5 only for

- print output from jobs in interactive partitions
- print output from jobs which have been submitted to batch partitions
- displaying the line command area to the right of the data line

(your administrator must have requested an alternate screensize in the CICS/VS Terminal Control Table DFHTCT and Program Control Table DFHPCT). For wide screen support, parameters C1 and C2 both range from 1-132. The default settings are 1 for C1 and 132 for C2.

Print output from an interactive partition is placed in the print area and then formatted for the wide screen when it is displayed. The output can also be saved in your library and then displayed using the /LIST or /DISPLAY commands, or via the /LISTP command for output from a batch partition.

The wide screen is also used if you retrieve output with the GETL procedure and place it in your print area, from where it is displayed automatically. Otherwise you can save the output in a library member using the GETL procedure with the PRINT parameter, and later display it with the /LIST or /DISPLAY command.

ERASE or NOERASE feature allows you to set the VSE/ICCF full screen erase feature ON or OFF. Normally, the 3270 screen is erased each time any new data is written to the screen. Specifying the NOERASE operand causes only the lines receiving new data to be erased before data is written to the screen. Specifying the ERASE operand returns the terminal to the standard screen erase setting.

When you work at an IBM 5550 with 3270 Emulation and when either of the following two conditions hold, all data on the screen is erased even though the NOERASE operand was specified:

- The screen window has been reduced to less than 80 columns with the /SET SCREEN command.
- Data (e.g., print data or SYSLOG data) from an interactive partition execution other than full screen writes (DTSWRTRD) is displayed on the screen.

When switching between two screen output areas, you should first establish a NOERASE environment so that data written to one area of the screen will not be erased when data is being written to another area of the screen. (See the example below.)

ALARM or NOALARM feature allows you to set the audible alarm for your terminal ON and OFF. If ALARM is specified, and if the terminal has the audible alarm facility, a 'beep' will occur each time something is written to the screen except when hardcopy mode is set.

CLEAR feature causes the active output area of the screen to be erased. This feature is most useful when NOERASE has been specified. When the CLEAR operand is used, only the active output area on the screen is cleared. (See the example below.)

The numeric parameter features (i, d1, d2, c1, c2) allow you to vary the size of the input area of the screen ('i' operand), the screen lines constituting the output area ('d1' and 'd2') and/or the screen columns constituting the output area ('c1' and 'c2'). If any one of these operands is specified, all preceding operands in the series must be specified. If one or more operands are not specified their present settings will not be changed unless one of the operands specified causes an inconsistency with one of the operands not specified.

The 'i' parameter controls the number of screen lines available for input. The normal screen consists of one 80 character input line, followed by a scale line followed by a display (or output) area containing a certain number of output lines depending on screen size. The 'i' parameter can be specified as 2, 3 or 4 to increase or 1, 2, or 3 to decrease the size of the input area. You may want to increase the size of the input area so that you can enter more commands or data lines in a single terminal transmission. For example, if the typical command or data line length (using tabbing) is 15 to 20 characters, having an input area of four lines would allow you to enter up to 16 or 18 command or data lines (separated by logical line end characters) in a single terminal transmission.

If the 'i' parameter is the only parameter specified and if it is increased beyond its current setting and if the current output area immediately follows the input area, the output area size and location will automatically be adjusted. If 'i' is decreased, the output area size and location will not be adjusted.

The 'd1' and 'd2' parameters control the vertical size and location of the output area. The normal output area on a 22 line screen, assuming a one line input area followed by a scale line, is screen line number 3 to 22. Thus, the default settings for d1 and d2 are 3 and 22. To vary the area of the screen to which output will be directed, specify 'd1' as the new starting screen line number and specify 'd2' as the number of lines in the area. If the starting line number specification would cause the input area or scale line to be overlapped, the 'd1' factor will be adjusted forward to the line following the scale line.

Similarly, the 'c1' and 'c2' parameters control the horizontal location and size or width of the output area. The normal screen width is 80 columns. However, the output area can be specified as starting at any given screen column location and can continue for any given number of columns. For example, the output area could be set to occupy columns 41 through 80 of the screen. All output which normally would have appeared starting at column 1 would now appear at column 41. Lines exceeding 40 characters would be truncated accordingly.

COLUMN feature allows you to set the 'c1' and 'c2' parameters without having to specify the three previous parameters (i, d1, d2). If only one parameter is specified, this is assumed to be 'c1'; 'c2' will be assumed to be 80. If 'c1' and 'c2' are omitted or if RESET is specified, the horizontal output area will be set to column 1 through column 80.

Note: Do not set your screen width to 1, because this will not accommodate meaningful output.

ROW feature allows you to specify the vertical location of the screen relative to the end of the input area of the screen. The 'r1' and 'r2' operands specify the beginning and ending row number which you want to be the output area. Since the row numbers specified are relative to the end of the input area, you need not be concerned with the present number of lines in the input area.

If 'r1' and 'r2' are omitted, or if RESET is specified, the vertical span of the output area is set to the maximum available area based on the input area size. If only one parameter is specified, this is assumed to be 'r1'; 'r2' will then be based on the specification of 'r1'. If 'r1' and 'r2' are specified, 'r2' must be 4 higher than 'r1'.

The RESET operand in all cases resets the features to the system screen default values.

This command gives you a form of split-screen operation as shown in the example below. Your terminal should be equipped with PF keys to enable you to move easily from one area to another.

Example:

To define two active portions of the screen so that you can use one area while retaining information in another, first set three PF keys as follows:

```
/set  pf1  /set  screen  row  1   10
/set  pf2  /set  screen  row  11  20
/set  pf3  /set  screen  clear
```

These settings will allow you to vary your output area between the top and the bottom half of the screen by pressing PF keys 1 or 2. To clear the screen output area, press PF3.

Next, set the NOERASE screen feature so that a write to one area of the screen will not erase the entire screen.

```
/set  screen  noerase
```

Now you can use two sections of the screen: one to retain information while you are actively using the other.

Suppose you want to purge unneeded members from your library. You could place a library display in one area of the screen and retain it there while you use the other area for the purges and lists of library members.

Pressing the PF2 key sets you to the second screen area. Now enter the /LIB command. If the display will not fit entirely into the lower screen area, the scale line will indicate that you are in list mode (LS). Entering the /CANCEL command (or pressing the Cancel key) will return you to command mode.

Now press the PF1 key to set the output area to the top portion of the screen. You can now do /LIST, /PURGE or other functions while retaining the library display in the lower portion of the screen.

You can also page the display in the top portion of the screen forward by pressing the PA2 key.

/SETIME Command

The /SETIME command alters any of the three time factors that can affect terminal activity or interactive partition execution.

```
/SETIme        [EXEC]  m [n]
               TIMEOUT t
```

m is a number from 0 to 32767. It represents the maximum number of execution units that the next or current job can accumulate without being automatically canceled.

n is a number from 0 to 65535 which represents the maximum total time in seconds that a given job can run in an interactive partition.

t is a number from 60 to 3600 which represents the number of seconds of terminal inactivity that will constitute a timeout.

The /SETIME command is effective in command and execution mode.

The three time factors which can be set are:

1. Total interactive partition execution units: expiration of these units will force the job to be canceled. (An execution unit is approximately one second of elapsed (wall clock) time during which the interactive partition is eligible for execution cycles.)

2. Total time in the interactive partition: expiration of this time causes the job in the interactive partition to be canceled.

3. Total time between instances of terminal activity: when this time expires, log-off is forced.

To temporarily override the number of execution units which a job can accumulate before it is canceled, specify the command as /SETIME EXEC m or merely /SETIME m. (The EXEC operand will be assumed if omitted.) If the n operand is omitted, it will be assumed to be four times the m value unless your default for n is larger than four times m.

To temporarily override the number of execution units and total interactive partition time which a job can accumulate before it is automatically canceled, specify the command as /SETIME EXEC m n or merely /SETIME m n.

After the current or next execution has completed, the limits for the number of execution units, and for interactive partition time, will be reset to your default values.

This command allows you to exceed your user profile specified maximum execution units for any one job.

You should not set unnecessarily large limits; this would give you and the system little protection. Time limits protect you and the system if the job enters a loop. The job would be canceled, and the storage freed, when the time limit had been reached.

The execution units and total interactive partition time limits can also be set for a given job in the job stream itself. See the TIME= operand of the /OPTION job entry statement.

The /SETIME command followed by the TIMEOUT operand lets you vary the terminal inactivity timeout value. This value can be set as low as 60 (seconds) and as high as 3600. Whatever limit is set, you will be automatically logged off if no terminal activity occurs within the specified limit, unless your user profile indicates that you should not be forcibly logged off due to a timeout condition. (The timeout-forced logoff does not apply when a user is in execution mode.)

Examples:

```
*READY
/setime 300
*LIMIT HAS BEEN SET
*READY
/setime 500 5000
*LIMIT HAS BEEN SET
*READY
/setime timeout 1800
*LIMIT HAS BEEN SET
*READY
```

/SHIFT Command

The SHIFT command causes the display on a screen to move left or right.

```
/SHIft        {RIght (nnn)|LEft (nnn)|OFf}
```

RIGHT moves the screen window to the right by 'nnn' columns. This enables you to view the righthand part of a record which is currently not visible on the screen. The data being displayed moves to the left.

LEFT moves the screen window to the left by 'nnn' columns. The data being displayed moves to the right.

OFF causes a shift left to column 1 and a display in 'wrap-around format' (also known as '80-character format'). When the display has this format, pressing ENTER causes the next screen to be displayed in the same format. The default PF key settings in list and spool mode (see Figure 2-1 on page 2-12) will be reestablished.

The /SHIFT command is valid in LS (list), SP (spool print), and RD (conversational read) mode. It controls shifting of *print-type data*, that is, VSE/POWER list output, print-type members (members whose name ends with '.P'), and the print area $$PRINT. When issued for normal VSE/ICCF library members or for print-type members that are compressed or displayed in hexadecimal format, the /SHIFT command is ignored and treated as if ENTER had been pressed.

/SHIFT LEFT or RIGHT causes a display in printline format (as opposed to wrap-around or 80-character format), that is, part of the line may be invisible. It also suppresses all the default PF key settings for list and spool mode. When your display has printline format, SYSLOG messages, too, have a portion of the message line truncated off.

The maximum value that can be specified for 'nnn' is 100. The left margin of the screen (the window) cannot be moved to a position higher than column 100 of the data, and not lower than column 1. The scale line moves by the same amount as the data. While your display is in printline format, pressing ENTER causes the next screen to be displayed in printline format with the same shift amount as the previous screen.

If the /SHIFT command is entered more than once, the 'nnn' values are accumulated. For example, entering /SHIFT RIGHT (30) twice would move the left margin of the screen to column 60 of the data. Again, the left margin of the window can move up to column 100 and down to column 1 at the most.

[/]SHOW Command

The [/]SHOW command displays the settings of options, features and parameters affecting your terminal. The [/]STATUS command has the same function.

```
[/]SHow        BS
[/]STatus      BUFfer
               BYPass
               CASe
               CHar
               CLAss
               COMlib
               CTL
               DATAnl
               DAte
               DEL
               DELAy
               END
               ESC
               EXec
               HEX
               IMPex
               LIMit
               LIBrary
               LINesize
               LOG
               MOde
               MSG
               PF[ED][EX][LS]
               PFnn[ED][EX][LS]
               PF SAVE
               PSize
               SCreen
               TABChr
               TABs
               TErmid
               TIme
               USer
               VERify
               XLate
```

The '/SHOW' format is effective in any mode except edit mode; the 'SHOW' format is effective only in edit mode.

The /SHOW command displays the status of options set by the /SET command, or which were set when you logged on. These options are:

Operand:	Display Setting Of:
BS	Backspace character
BUFFER	Size of the print area
BYPASS	Control character bypass
CASE	Input/output translation
CHAR	Any control character
CLASS	Your execution class
COMLIB	Common library search
CTL	Any control character
DATANL	Data analysis feature
DATE	Date and time

Operand:	Display Setting Of:
DEL	Delete character
DELAY	Multi-command input delay
END	Logical line end character
ESC	Escape character
EXEC	Status of job in an interactive partition
HEX	Hex control character
IMPEX	Implied execute
LIMIT	Various time limits
LIBRARY	Your primary and connected libraries
LINESIZE	Line width
LOG	Type of terminal logging
MODE	Terminal mode
MSG	Message mode
PF[ED][EX][LS]	Set PF keys in CM, ED, EX and LS modes
PFnn[ED][EX][LS]	Set PFnn keys in CM, ED, EX and LS modes
PF SAVE	Content of PF-key save area
PSIZE	Default partition size
SCREEN	Screen attributes, size and location
TABCHR	Tab character
TABS	Tab location
TERMID	Your terminal id
TIME	Various time limits
USER	Your four character userid
VERIFY	Input verify
XLATE	Input/output translation

If no operands are specified, DATE, TERM, USER, PSIZE, LIBRARY and MODE and the output of the /USERS command will be included in a single display. If an operand is invalid, the default display will consist of the IMPEX, LOG, PF, VERIFY, LINESIZE, BYPASS, DATANL and COMLIB settings.

Example:

```
*READY
/status
DATE=09/02/83 TIME=01/30/00 EXECUTION UNITS=0030
USER ID CODE  = USRA
TERMINAL CODE = L2D3
LIBS: MAIN = 0001 CONN = 0005 COMMON = 0002
SYSTEM MODE = Command
DEFAULT PARTITION SIZE=0192K
NUMBER OF TIME SHARING USERS LOGGED ON = 0003
*READY

/show tabs
TABS = 10 16 36 72
*READY

/show pf
01 /SET SCREEN ROW 1 10
02 /SET SCREEN ROW 11 20
03 /SET SCREEN CLEAR
*PROGRAM FUNCTIONS SET OFF
*END PF DISPLAY
*READY

/show char                (All control character assignments are shown
D T B E E H                in vertical order.)
E A S N S E
L B   D C X

¬ ; < | % #
5 5 4 4 6 7
F E C F C B
*READY
```

/SKIP Command

The /SKIP command moves the display forward or backward in the data being viewed. At the conclusion of the skip, displaying automatically resumes with the next screen of data.

```
/SKip          [m|-m|S+n|S-n|P+k|P-k|NExt|PRevious|CUrrent|TOP|BOTtom|END]
```

m	is a number from 0 to 99999 representing the number of logical lines (from the last line displayed) to be skipped over during a display. -m causes backward skipping.
S+n|S-n	is a number from 0 to 99999 representing the number of logical lines (from the top of the screen) to be skipped over during a display to a 3270 terminal. -n causes backward skipping.
P+k|P-k	is a number from 0 to 99999 representing the number of execution mode printer pages (skips to carriage channel 1 or top of form) to be skipped over during a display of print output. -k causes backward skipping.
NEXT|PREVIOUS	causes a scroll forward or backward by one screen. The unit of scrolling is the size of one logical screen, which is defined through the /SET SCREEN command.
CURRENT	causes a (re)display of the current page.
TOP	displays the top of the file, which can be a VSE/POWER list file for /LISTP processing or a VSE/ICCF member for /LIST processing.
BOTTOM|END	displays the end of the file present when using the /LIST or /LISTP command.

/SKIP is effective in SP (spool print) or LS (list) mode during a display of print output.

Your terminal is placed into list mode when you enter commands that display library members or work areas, for example with /LIST and /DISPLAY. During editing you also enter list mode with the PRINT and PRINTF editing commands. The same is true when you use the /LISTP command to display VSE/POWER spool file output. List mode is also entered when an interactive partition execution displays print output.

When m or -m is specified, m lines of printout are skipped over counting from the last logical line displayed. (During execution mode output displayed on a 3270, a logical line can consist of two physical lines.)

When S+n or S-n are specified, n lines of printout are skipped beginning at the first line of the last screen of data printed or displayed. This form of the command is most useful for 3270 terminals in that it allows the screen to be scrolled forward (S+n) or backward (S-n) by n lines. For example, PF keys might be set to /SKIP S-5 and /SKIP S+5. Then each time one of these PF keys is pressed, the screen retreats or advances 5 lines.

The P-k and P+k operands can only be used when you are displaying interactive partition job output. k represents the number of print pages to be skipped. A print page is indicated by an advance to the top of form in the print output of the running program. If one of these operands is entered but the program did not issue any advances to the top of the form, the skip will proceed to either the beginning or the end of the print area. The default is m=0.

The NEXT and PREVIOUS operands let you scroll through the print output one (logical) screen at a time. These operands are easier to use than S+n or S-n where you would have to remember the logical screen size to specify 'n'. The operands are valid for the display of print-type and normal VSE/ICCF library members. You are encouraged to set PF keys equal to /SKIP NEXT and /SKIP PREVIOUS.

/SKIP CURRENT causes a redisplay of the current page. This can be handy if you temporarily leave the current display, for example, by issuing

```
-   the /MSG command to see if any messages arrived, or
-   the /SET PFnn command to set a PF key.
```

BOTTOM or END advances the display to the end of the area being displayed, whereupon a message is issued. By pressing ENTER, the command is finished and you return to command mode.

Notes:

1. *When a /LIST command is issued for a compressed member, only forward skips are allowed and it must be a SKIP to a line that is not yet displayed on the screen. For example, if six logical lines are displayed on the screen, /SKIP S+5 will be rejected.*

2. *The P-k and P+k versions of the /SKIP command are not effective when displaying VSE/POWER spooled output.*

3. *For print-type members specified in one of the /LIST commands, the parameters 'm' and 'n', as well as the line number printed in front of each line by the /DISPLAY command, relate to lines in print format and not to the records of the member. Thus two records of a member can account for one line of print output.*

4. *The same is true if /SKIP has been entered in list mode and the output is displayed in print format. The number of lines to be skipped relates to lines in print format, as is also the case if /SKIP is issued in execution mode.*

Example:

```
*READY
/display counts
0001 CARD 1
0002 CARD 2
0003 CARD 3
0004 CARD 4
 (end of print area)
/sk 200
0205 CARD 205
0206 CARD 206
 (etc.)
```

SORT Procedure

The SORT procedure sorts a library member according to the EBCDIC sequence of hex-values and either saves the output as another library member, places it in the punch area, writes it back to the terminal or uses it to replace the original member.

```
SORT        name1 [pass] [SEQ seqinf] ┌ PUNCH name2 ┐
                                      │ PUNCH  *    │
                                      │ PRINT       │
                                      └             ┘
```

name1	is the library member to be sorted.
pass	is a 1 to 4 character password which must be specified for password protected members.
SEQ seqinf	is the information indicating where the sort sequence fields are located. The format is XYYZZXYYZZ...where: 'X' is A or D indicating ascending or descending sequence, 'YY' is the starting column number in the record, 'ZZ' is the number of columns in the sequence field. Up to four sequence fields may be specified. If the SEQ operand is omitted, the sort will be on columns 1 to 15.
PUNCH name2 PUNCH * PRINT	defines where the sorted output is to be saved: into a library member (name2), into the punch area $$PUNCH (*) or that it is to be displayed only at the terminal (PRINT).

Note: Because / signals end-of-data to the sort program, members must not have /* as data in columns one and two.*

Examples:

1. Sort a member into sequence and display the result at terminal:

   ```
   sort dtamem seq a0105d6704 print
   ```

2. Sort a member into default sequence and have it replace itself:

   ```
   sort memba
   ```

/SP Command (See /STATUSP Command)

$SPACE Procedure

The $SPACE procedure causes the DTSSPACE program to be loaded into an interactive partition and run.

```
$SPACE
```

This program looks at each dynamic disk space allocation area within the system and prints its status. The information printed includes the volume and location of the space area, individual allocations within it and an indication of the percentage of the area which is free.

Example:

```
*READY
$space
*RUN REQUEST SCHEDULED FOR CLASS=A
-----------------------------------------------------------------------------
        DYNAMIC SPACE ALLOCATION MAP -- VOLUME # 01
BEG=456   FOR 38      DEV=3330  SYS001 SER=SYSWRK COLD AREA COLD STARTED
-----------------------------------------------------------------------------
                  FILE ID                   FROM      FOR    OWNER  DISP
DYNAMIC-SPACE-CONTROL-INFORMATION                 456        1  SYS  KEEP
 .
***TOTAL*** ALLOCATED=0001 FREE=0037% FREE=97%
 .
-----------------------------------------------------------------------------
        DYNAMIC SPACE ALLOCATION MAP -- VOLUME # 02
BEG=1653  FOR 76      DEV=3330  SYS001 SER=SYSWRK WARM AREA WARM STARTED
-----------------------------------------------------------------------------
                  FILE ID                   FROM      FOR    OWNER  DISP
DYNAMIC-SPACE-CONTROL-INFORMATION                1653        1  SYS  KEEP
IJSYS02.AAAA.T264                                1654       50  ***  ****
 .
***TOTAL*** ALLOCATED=0051 FREE=0025% FREE=32%
 .
*PARTIAL END PRINT
```

/SQUEEZE Command

The /SQUEEZE command converts a library member from display to compressed format. It is only valid in command mode.

```
/SQueeze      name [password] [SAVEIN] [LOWER]
```

name	is the name of a library member in display format which is to be compressed.
password	is only required if the library member being squeezed is password protected.
SAVEIN	causes the display format version of the member to be saved in your input area (destroying any previous contents of the input area).
LOWER	is specified when the member being compressed contains more lower case than upper case alphabetic data.

Library members are normally retained in display format so that certain VSE/ICCF operations (such as edit) can be performed on them. However, once a library member has reached a finished or an inactive state, it can be compressed to save disk space.

A compressed member usually takes up 55 to 70 percent less disk space than a member in display mode. The actual savings will depend on the number of blanks in the data and the data itself. If the data consists mainly of characters from the 64 character EBCDIC graphic subset and is not very dense, the saving will be greatest.

The compressed member replaces the display copy of the member in the library. If you want to compress a library member and yet retain the display version of it, enter the SAVEIN operand. Then, to save the display format version of the member, issue the /SAVE command after the /SQUEEZE command. If the SAVEIN operand is used in a /SQUEEZE command issued from a procedure, a /SAVE command should be issued in the procedure following the /SQUEEZE; otherwise, the input area will be lost.

Once a library member has been compressed, it can be converted back to display format by entering input mode (/INPUT command) and issuing the /INSERT command for the member. The /LIST command can be used to list a library member which is in compressed format. The /SKIP command can be used to skip forward while displaying a compressed member, but not backward. Note that the number of records saved by compressing a library member is credited to the user who issued the /SQUEEZE command. The SAVEIN operand cannot be used if you are in asynchronous execution mode (see /ASYNCH command).

Note: Print-type members that have been processed by the /SQUEEZE command are displayed in 80 character record format, not in print format.

Example:

```
*READY
/squeeze payprog savein
*INPUT RECS = 1230 OUTPUT RECS = 0467 SAVINGS = 62 PERCENT
*END COMPRESS
*READY
/save payproga   (save display format version)
*SAVED
*READY
```

[/]STATUS (See /SHOW Command)

/STATUSP Command

The /STATUSP command displays the status of jobs submitted to VSE/POWER to be run in a VSE batch partition, or for transmission to another node. The /STATUSP command can only be used in command mode.

```
/STATUSP       jobname [jobnumber]
/SP
```

jobname is the 2 to 8 character name of the VSE/POWER job whose status is to be displayed.

jobnumber specifies the 1 to 5 digit jobnumber assigned to the job by VSE/POWER. This parameter must be specified if jobname is not unique. If you fail to do so, the first queue entry found will be taken.

The status is displayed as:

```
    *STATUS = xy - job status
```

The 'xy' field (which is supplied by VSE/POWER) indicates that the job:

```
x =  N  cannot be located.
     R  is in the reader queue.
     L  is in the list queue.
     P  is in the punch queue.
     X  is in the transmission queue.
y =  *  The job is running. Otherwise, this is the
        disposition of the output of the job.
```

The 'status' field indicates whether the job is:

- NOT FOUND, the named job is not known to VSE/POWER.
- COMPLETED, the named job has completed execution and can be displayed using the /LISTP command. The list/punch output can be retrieved via the GETL/GETP procedure.
- EXECUTING, the job is running.
- AWAITING EXEC, the job is in a queue awaiting execution in a VSE partition.

Notes:

1. *The output displayed by the /STATUSP command may also include VSE/POWER messages, which are identified by the characters '1Q', '1R' and '1V' and are followed by a message code. These messages are explained in VSE/POWER Messages, SH12-5520.*

2. *The /STATUSP command cannot be used within a procedure.*

3. *If you are using an IBM 3270 you can set a PF key to part of a command, for example '/STATUSP' (don't forget the blank after /STATUSP), which allows you to type in the jobname and jobnumber and press the PF key to display the status of the job.*

Example:

Display the status of MYJOB, whose jobnumber is 00024:

```
/sp  myjob  00024
*STATUS = LL - JOB COMPLETED
```

STORE Macro

The STORE macro saves the contents of the punch area in the library under the specified name.

```
STORE          name [pass]
```

name is the library member name under which the contents of the punch area will be saved.

pass is a four character password which needs to be specified if the member is to be password protected.

The STORE macro must be issued immediately after the execution that produced the punch output. No other execution should be allowed to take place, even if it does not write data to the punch area.

STORE can be used to store data placed in the stack during an edit operation since the editor stack area and the execution punch area are the same area. The STORE macro copies the punch area to the input area and executes a /SAVE command. If the message 'NAME IN LIBRARY' appears, you must first issue the /SAVE or /REPLACE command for the data that is now in the input area.

SUBMIT Procedure

The SUBMIT procedure submits VSE/ICCF or VSE job streams for execution in another VSE partition.

```
                              ┌          ┐
SUBMIT       name [pass1]  │DIRECT │  [PRINT] [PWD=pass2] [ICCFSLI]
                              │RETURN │
                              └          ┘
```

name	is the name of a VSE/ICCF library member consisting of VSE job control language or VSE/ICCF job entry statements which together are submitted as a job stream for batch execution via VSE/POWER in another VSE partition. Certain names should be avoided; refer to the section "Submit-to-Batch Capability" on page 9-29.
pass1	is a four character password which need only be specified if the library member is password protected.
<u>DIRECT</u> RETURN	If DIRECT is specified, the output from the job will be directed to the printer or punch unit. If RETURN is specified, the output will be placed in the VSE/POWER output queues. You can view this output using the /LISTP command, retrieve it with the GETL or GETP procedures, and finally dispose of the output using the /ERASEP command. However, only the submitter or the user designated in the DEST parameter of the VSE/POWER JECL statement * $$ LST may access an output queue entry. An authorized user may specify ANY in the DEST parameter (authorization is established in the VSE/ICCF user profile). This makes the output accessible to any user.
PRINT	The PRINT operand causes all VSE job control language which is passed to VSE/POWER to be displayed.
PWD=pass2	is a 1 to 8 character password which is placed in the VSE/POWER $$ JOB statement to password protect the output. If PWD=pass2 is specified, it must also be specified in the GETL, GETP, or GETR procedures to retrieve VSE/POWER queue entries. This parameter can be specified in any parameter location.
ICCFSLI	If ICCFSLI is not specified, the VSE/ICCF member that is to be submitted to the VSE/POWER reader queue, will be read from the VSE/ICCF library file and written to the VSE/POWER queue.

If ICCFSLI is specified, a * $$ SLI statement is generated. The data will not be written to the VSE/POWER queue. Instead, at execution time, VSE/POWER itself will read the data from the VSE/ICCF library file. Be aware that * $$ JOB and * $$ EOJ statements in the library member will be ignored; in this case, the * $$ JOB statement that SUBMIT produces will be in effect. Also be aware that a * $$ LST and/or a * $$ PUN statement will cause VSE/POWER to discard the original jobnumber and to assign a new one.

When retrieving the member from the VSE/ICCF library file, VSE/ICCF uses the search chain of libraries (primary, connected plus common library) that is valid at the time of submitting.

/SUMRY Command

The /SUMRY command produces a summary listing of a library member. A summary listing includes any line beginning with a slash (/), plus a count of all lines which do not begin with a slash. The command is only effective in command mode and can only be used for non-compressed members.

```
/SUMry        name [password]
```

name	is the name of the library member to be summarized.
password	is the member's four character password; required only if the member is password protected.

Example:

```
*READY
/sumry fortprog
0001 /LOAD VFORTRAN
0002 /OPTION TRUNC
CARD IMAGE COUNT = 0009
0012 /DATA
CARD IMAGE COUNT = 0005
*END PRINT
*READY
```

/SWITCH Command

The /SWITCH command is used to switch from one library to another. It is valid in command mode.

```
/SWitch        lib-no|LIBs|RESet|OFF
```

lib-no	is the number of the library which you want to become your primary library.
LIBS	causes the connected library to become the primary and the primary the secondary library.
RESET|OFF	causes the library configuration to be as it is in your user profile.

The /SWITCH command causes the library whose identification is indicated by lib-no to become your current primary library. All later requests to access, update or save library members will cause the directory of the new library to be scanned. lib-no can be any library which you are authorized to use. That is, it can be your default library, one of the alternate libraries specified in your user profile, or a public library. The /USER LIBRARY command will tell you which private libraries you can access. The /LIBRARY and /LIBRARY CONN commands will display the library identifications of the current primary and connected libraries.

You can only switch to a public library, or to a private library that you are authorized to use (does not apply to the VSE/ICCF administrator). If the /SWITCH command is used in a procedure it will only be on a temporary basis. In some situations the /SWITCH command has no effect. For example, if the specified library number does not exist, or if you are not authorized to use it. /SWITCH also has no effect if the library is already in use, or if it is a connected library.

If you have used the /CONNECT command to connect a secondary library to the primary library, you may want to change the search order of the two libraries. Issue the command with the LIBS command, which makes the primary library into the secondary library, and the secondary library into the primary.

If the RESET operand is issued, your main library as indicated in your user profile becomes your primary library. If any other library is connected at the time the command is issued, this logical connection is terminated, i.e. RESET restores the library configuration to user profile status.

Example:

```
*READY
/switch 12
*SWITCHED
*READY
/input
aaaaa
/end
*READY
/save tmemb
*SAVED
*READY
/switch 14
*SWITCHED
*READY
/list tmemb
*FILE NOT IN LIB
*READY
```

/SYNCH Command

The /SYNCH command is used to resynchronize interactive partition execution with your terminal. The use of this command assumes that you had previously started a job in an interactive partition then disconnected the execution from your terminal using the /ASYNCH command.

```
/SYNch
```

This command is only effective in command mode.

While in asynchronous execution mode you can perform any normal VSE/ICCF command function (such as editing) other than start another job execution. When you want to see the result of your execution, issue the /SYNCH command to place your terminal back into execution mode. The /SHOW EXEC command can be used to display the status of your job, and to tell whether the /SYNCH command is required.

Example:

```
*READY
/exec payprog
*RUN REQUEST SCHEDULED
/asynch
*ASYNCHRONOUS EXECUTION MODE ENTERED
/edit artest
   .
   .  (edit commands)
   .
quit
END EDIT
*READY
/synch
   .
   .  (job output)
   .
```

[/]TABSET Command

The [/]TABSET command lets you establish your own logical tab settings for a line. These settings determine the number of spaces to be inserted in the line when either the logical Tab character or the tab key is used.

```
[/]TABset      n1...nN|language|OFF|CLEAR
```

n1...nN	are column positions for logical tab settings.
language	is the programming language or other character identification used to represent preset tab values.
OFF|CLEAR	causes all logical tab settings to be cleared so that tabbing is not possible.

The '/TABSET' form of this command is effective in command and input mode, while the 'TABSET' form is effective only in edit mode.

The [/]TABSET request can be followed by from one to eleven one-or-two digit numbers that do not exceed the value of 80. If more than eleven numbers are specified, only the first eleven are used.

Each [/]TABSET request overrides all previous /TABSET commands. Tab settings can be displayed with the /SHOW TABS command.

Your tab settings will remain in effect until another /TABSET command, or edit TABSET command, is issued.

Tab characters in DBCS members are interpreted as tab characters, only if the following two conditions hold:

- The tab characters do not appear in a string that has a double-byte representation.
- The tab characters are included in a *new* line, that is, a line which has just been created or inserted in edit mode.

You can set certain predefined tab values. Below is a table of the keywords that are allowed, and the tab positions that they set:

Keyword Operand	Tab Positions
ASsembler	10,16,36,72,73
BASic	10,20,30,40,50,60
COBol	8,12,16,20,24,28,32,36,40,44,73
FORTran	7,73
PL1	5,10,15,20,25,30,35,40,45,50
PLI	(same)
RPG	6,20,30,40,50,60
TENS	10,20,30,40,50,60,70

Only the capitalized portion of the keyword need be specified as the operand.

Example:

```
*READY
/input
/tabset cobol
*TABS SET
/set tab=;
*CONTROL SET
/load fcobol
;id;division.
;program-id. jactest.
;environment division.
;data division.
;working-storage section.
;77;inarea pic x(20).
;procedure division.
;;display ' please enter your name'.
;;accept inarea from console.
;;display ' hello ' inarea.
;;stop run.
/end
*READY
/list
/LOAD FCOBOL
       ID  DIVISION.
       PROGRAM-ID. JACTEST.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  INAREA PIC X(20).
       PROCEDURE DIVISION.
           DISPLAY ' PLEASE ENTER YOUR NAME'.
           ACCEPT INAREA FROM CONSOLE.
           DISPLAY ' HELLO ' INAREA.
           STOP RUN.
*END PRINT
*READY
```

/TIME Command

The /TIME command displays the date and time and the number of background execution units.

```
/Time
```

This command is valid in command and execution mode.

If your terminal is in command mode, the number of execution units displayed is the number accumulated by the last background job that was run. If the /TIME command is entered while the background job is running, the execution units displayed are those that the job has accumulated up to the time when the command was entered.

Note: The date can appear in the form Day, Month, Year (DMY), or Month, Day, Year (MDY), depending on the format active when VSE/ICCF was started.

Example:

```
*READY
/time
DATE=09/02/83 TIME=01/30/55 EXEC UNITS=0024 PARTN TIME=00120
*READY
```

/USERS Command

The USERS command displays various fields in your user profile, and the libraries to which you have access. Without an operand, the /USERS command displays the total number of terminal users who are presently logged on. It is effective in any mode except edit and message mode.

```
/USers        [PROfile|STAts|LIBrary]
```

PROfile displays various defaults in your user profile, such as userid, maximum statements per input area, and your execution time limit (see Example 2).

STAts displays statistics in your user profile such as total logons, keyboard and run requests (/RUN or /EXEC). It also shows the execution units used, your total logon time, and disk space. The statistics cover the period since the last installation accounting run was made.

LIBrary displays the libraries to which you have access. This includes your main library, the common library, and any alternate libraries to which you can switch.

Examples:

1. Display the number of users logged on:

```
/users
NUMBER OF TIME SHARING USERS LOGGED ON = 0003
*READY
```

2. Display the defaults in my user profile:

```
/users profile
USER LIB MAXST MAXPU MAXPR TIMLM TIMOT PPTIM LNSIZ DEFLTS PROC
bbbb  4   350   2000 2000   900   600   3600  80    ¢#<>!% start
*READY
```

Notes to Example 2

```
USER   your user identification
LIB    your default main library
MAXST  size of input area (statements)
MAXPU  size of punch area (statements)
MAXPR  size of print area (statements)
TIMLM  default time limit per execution
TIMOT  time-out value
PPTIM  maximum time in interactive partition before cancel
LNSIZ  line size for /LIST and /DISPLAY commands
DEFLTS delete, tab, backspace, line-end, escape and hex characters
PROC   logon procedure
```

VSBASIC Procedure

The VSBASIC procedure is used to compile and run a VS BASIC source program. Alternately, this procedure may be used to compile a program and punch an object deck or to load and run an object deck.

```
VSBASIC     [LOAD  ]  name1 [OBJ name2] [JES name3] [DATA name4] [RUN]
            [SOURCE]             *           *            *
```

LOAD	is specified if the member (name1) is a VS BASIC object program.
SOURCE	is specified or the operand is omitted if the member (name1) is a VS BASIC source program.
name1	is the name of a VSE/ICCF library member which contains the VS BASIC input in either source or object form.
OBJ name2	is the member name under which the VS BASIC object deck will be saved in the library. If the name already exists, the object module will replace the contents of the member. If 'OBJ *' is specified, the object deck will be placed in the punch area. If the 'OBJ name2' operand is omitted, no VS BASIC object deck will be produced.
JES name3	is the name of a member containing file definitions and/or other job entry statements for the execution of the program. 'JES *' indicates that you are to be prompted for the job entry statements.
DATA name4	is the name of a member containing the job stream (INPUT) data for the execution. If 'DATA *' is specified, the data will be read conversationally from the terminal as if /DATA INCON had been specified.
RUN	The RUN operand specifies that you are to be prompted for entry of a VS BASIC RUN OPTION statement.

If the first operand is specified as SOURCE, or is omitted, the named member is assumed to be a VS BASIC source program to be compiled and run. If the 'OBJ name2' operand is specified, an object deck will also be punched.

The RUN operand should not be specified if a RUN statement exists within the source program. If you are only compiling, and do not intend to run the program, specify the NOGO option on the RUN statement.

Examples:

1. Compile and run a VS BASIC program with conversational input:

```
vsbasic bascprog data *
```

2. Compile a VS BASIC program and save the object deck as library member 'BSCOBJ':

```
vsbasic bascprog obj bscobj
```

VSBRESEQ Procedure

The VSBRESEQ procedure is used to resequence a VS BASIC source program.

```
VSBRESEQ    name [incr|10 [pass]] [LIST] [NOUPD]
```

name	is the library member containing the VS BASIC program to be resequenced.
incr	is a decimal number from 1 to 5000 to be used as a resequencing increment. If omitted, the default increment is 10.
pass	is the 4 character password associated with the member if it is password protected.
LIST	displays the resequenced member.
NOUPD	generates error and warning messages; the resequenced member is not replaced in the library.

Examples:

1. Resequence a VS BASIC program:

   ```
   vsbreseq bascprog
   ```

2. Display a program in the resequenced form, generate any error or warning messages, but do not replace the member in the library:

   ```
   vsbreseq bascprog list noupd
   ```

Chapter 4. Editor

VSE/ICCF has two editors:

- A *full screen editor* for IBM 3270 and similar display terminals.
- A *context editor* which was designed for typewriter terminals and which must be used in procedures and macros and in the DTSBATCH utility.

The context editor works on one line at a time. This line is determined by locating a given string. In other words, the context editor works in the 'context' of that string.

The full screen editor, on the other hand, utilizes the entire screen. You can modify data simply by positioning the cursor at a point on the screen and typing the new data. Then, by entering a command or pressing a key, the modification is applied to the file.

This chapter describes the usage of the full screen editor (in the VSE/ICCF publications mostly referred to as 'the editor'). The context editor is described in Appendix B, "Context Editor" on page B-1.

This chapter consists of two major parts:

1. A discussion of some important editor applications.
2. A reference section giving detailed descriptions of all editor commands (and macros) in alphabetical order.

For an introduction to the basic editor functions, you should read and practice the information contained in the publication *VSE/ICCF Introduction to Interactive Programming*. Also note that the reference section in this chapter contains numerous examples that guide you into the usage of the editor commands.

Editor Characteristics and Capabilities

The editor allows you to create and modify files in the VSE/ICCF library from your display station. It utilizes all the facilities of the IBM 3270 Information Display System. In particular, it has the following capabilities:

- Creating a new library member while in edit mode.
- Split screen control, allowing multiple displays.
- Concurrent editing of several files in a single session.
- Concurrent editing of different areas within a given file.
- Formatting of the displayed data.
- Display or modification of data in hexadecimal or EBCDIC format.

- Moving and copying of data within one file or between two files.
- Entering of frequently used commands or data via PF keys.

The *file* being edited is either of the following:

- A member of the VSE/ICCF library.
- The input area (which you can save as a library member).
- The punch area.
- The print area.

Editing your print area is useful when, for example, you are looking for errors in a compiler listing. Having both the print area and your source program on one screen together may aid in finding an error.

Several files can be created, edited, saved in the library, and edited further without leaving edit mode. Split screen control allows displaying and editing multiple files,on one screen. And you can display and edit on one screen different areas of the same file.

The editor lets you move and copy data (contiguous or noncontiguous) from one or more places in a file to different places of the same or another file.

Note: *The editor does immediate updating. This implies that all changes are written to disk when you press ENTER or any PF key. Therefore, prior to applying "dangerous" changes to a file, you should, by using the COPYFILE macro, create a copy of the original under a new name; see also "Generation Member Group" on page 1-6 and "Maintaining Multiple Change Levels of a Member" on page 4-38.*

Editor Fundamentals

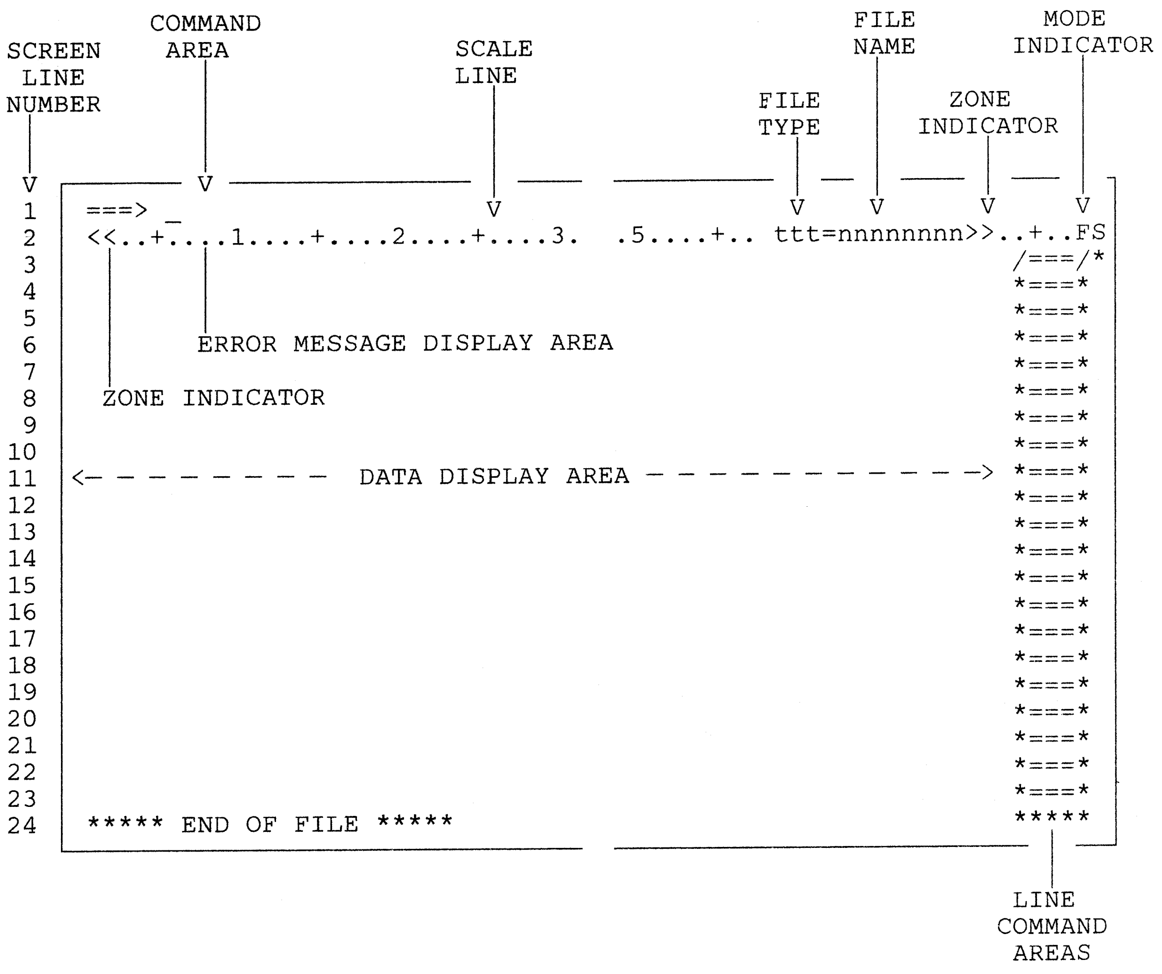
Screen Structure and Formatting

As a prerequisite to understanding the information presented later on, you should be familiar with the basic screen layout and the associated terminology, and with the concepts of *logical screens* and *format areas.*

Basic Screen Layout

The basic screen layout is the default layout which is displayed after invoking the editor via the ED macro. This screen layout is shown in Figure 4-1. The file being used for this example contains only blank lines.

You can deviate from the default layout by entering the VERIFY command with appropriate parameters.



```
 1   ===> _
 2   <<..+....1....+....2....+....3.   .5....+.. ttt=nnnnnnnn>>..+..FS
 3                                                               /===/*
 4                                                               *===*
 5                                                               *===*
 6                                                               *===*
 7                                                               *===*
 8                                                               *===*
 9                                                               *===*
10                                                               *===*
11  <- - - - - - - - -  DATA DISPLAY AREA - - - - - - - - - - - -> *===*
12                                                               *===*
13                                                               *===*
14                                                               *===*
15                                                               *===*
16                                                               *===*
17                                                               *===*
18                                                               *===*
19                                                               *===*
20                                                               *===*
21                                                               *===*
22                                                               *===*
23                                                               *===*
24   ***** END OF FILE *****                                     *****
```

Figure 4-1. Basic Screen Layout

The above figure shows the names of the various screen areas:

Command Area: Line 1 constitutes the command area. This area is identified by the characters '= = = >'. It normally consists of one physical screen line. Through the VERIFY command, you may expand this area to two, three, or four lines.

Any combination of editor commands separated by logical line end characters can be entered in the command area.

Scale Line: The scale line immediately follows the command area. It contains column indicators as an aid to working with fixed format data and displays the following information:

- A pair of two arrow heads designating the editing zone.
- Error message.

  Messages issued by the editor temporarily overlay the first 52 columns of the scale line. If no error message is displayed and a display offset is in effect (see the LEFT and RIGHT editor commands in the reference section of this chapter), the offset amount will be displayed in the error message display area.

- File type.

  The file type ('ttt' in Figure 4-1 on page 4-2) can be any of the following:

```
INP   - the input area.
MEM   - a library member, or the print area $$PRINT,
                          or the punch area $$PUNCH.
NEW   - a file being 'opened' via the ENTER command without
        operand, that is; a file being newly created.
```

- File name.

- Mode indicator.

  Throughout the edit session, this indicator appears as 'FS' meaning 'full screen' edit mode. If mixed data are edited, the FS indicator is preceded by 'DB'.

Data Display Area: This is the area where the individual records are displayed. Data displayed in this area of the screen can be modified simply by positioning the cursor and typing in the new data. This way, you can alter several lines before pressing ENTER.

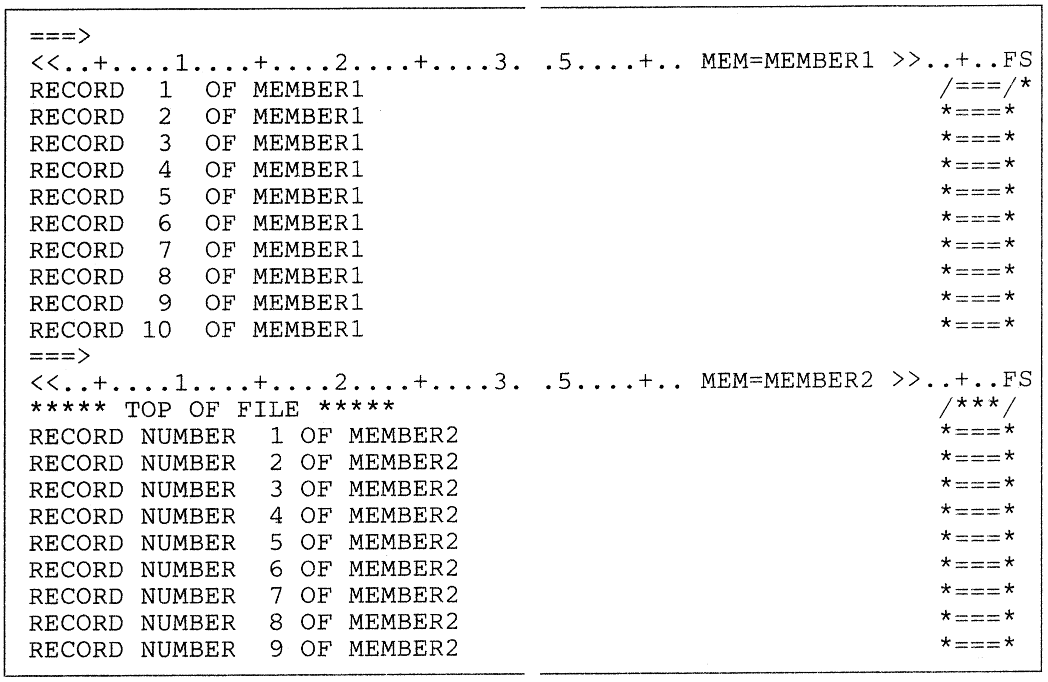
The *current line* record is identified through highlighting and, if the NUMBERS option is set off, the character string '/= = =/' in the line command area (see the next paragraph).

Line Command Area: Following each record displayed on the screen is a line command area. In this area, you can enter line commands to manipulate individual lines or groups of lines on the screen. Line commands are used to add, delete, duplicate, shift, move, or copy lines.

Line command areas are indicated by the character strings '*= = =*' or '/***/'. You enter line commands by overlaying these characters in the line command area. Section "Editor Line Commands" on page 4-142 presents more details.

Logical Screens

For the purpose of editing multiple files concurrently, the screen may be divided into (up to eight) *logical screens.* Each logical screen has its own command area and its own scale line. Each logical screen displays a different file. Figure 4-2 shows a screen which has been divided into two logical screens.



```
===>
<<..+....1....+....2....+....3.  .5....+.. MEM=MEMBER1 >>..+..FS
RECORD  1  OF MEMBER1                                       /===/*
RECORD  2  OF MEMBER1                                       *===*
RECORD  3  OF MEMBER1                                       *===*
RECORD  4  OF MEMBER1                                       *===*
RECORD  5  OF MEMBER1                                       *===*
RECORD  6  OF MEMBER1                                       *===*
RECORD  7  OF MEMBER1                                       *===*
RECORD  8  OF MEMBER1                                       *===*
RECORD  9  OF MEMBER1                                       *===*
RECORD 10  OF MEMBER1                                       *===*
===>
<<..+....1....+....2....+....3.  .5....+.. MEM=MEMBER2 >>..+..FS
***** TOP OF FILE *****                                     /***/
RECORD NUMBER  1 OF MEMBER2                                 *===*
RECORD NUMBER  2 OF MEMBER2                                 *===*
RECORD NUMBER  3 OF MEMBER2                                 *===*
RECORD NUMBER  4 OF MEMBER2                                 *===*
RECORD NUMBER  5 OF MEMBER2                                 *===*
RECORD NUMBER  6 OF MEMBER2                                 *===*
RECORD NUMBER  7 OF MEMBER2                                 *===*
RECORD NUMBER  8 OF MEMBER2                                 *===*
RECORD NUMBER  9 OF MEMBER2                                 *===*
```

Figure 4-2. Screen Layout with Two Logical Screens

Details about creating and using multiple logical screens will be provided in "Editing Two Logically Related Files" on page 4-24.

Format Areas

For the purpose of editing different portions of one file concurrently, a logical screen may be divided into (up to eight) *format areas.* Each format area has its own command area, but only the first format area in a logical screen has a scale line. Each format area can display a different portion of the same file. Figure 4-3 shows a screen which has been divided into two format areas.

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB ...+..F>
MAINPGM  CSECT                                                /===/*
         PRINT GEN                                            *===*
         BALR  5,0                                            *===*
         USING *,5                                            *===*
         OPEN FILOUT                                          *===*
LOOPA    EXCP  RDCCB                                          *===*
         WAIT  RDCCB                                          *===*
         CLC   R(3),=C'/* '                                   *===*
         BE    ENDCARD                                        *===*
         MVC   O(80,2),R                                      *===*
===> _
SAVEAREA DS    18F                                            /===/*
P        DC    CL120' '                                       *===*
R        DC    CL80' '                                        *===*
RDCCB    CCB   SYSIPT,RDCCW                                   *===*
PRCCB    CCB   SYSLST,PRCCW                                   *===*
RDCCW    CCW   2,R,0,80                                       *===*
PRCCW    CCW   9,P,0,120                                      *===*
IOA      DS    CL408                                          *===*
IOB      DS    CL408                                          *===*
FILIN    DTFSD IOAREA1=IOA,IOAREA2=IOB,IOREG=(2),           X *===*
               TYPEFLE=INPUT,BLKSIZE=400,RECSIZE=80,        X *===*
```

Figure 4-3. Screen Layout with Two Format Areas

Details about creating and using multiple format areas will be provided in "Editing Two Areas of One File Simultaneously" on page 4-23.

Types of Editor Commands

Editor Commands

You enter editor commands in the command area. These commands either control the editor operation (screen setup or processing options, for example), or they manipulate (locate, change, add, delete, copy, move etc.) data, starting at the current line.

Editor Line Commands

Line commands are, strictly speaking, editor commands, too. But they are special in several ways (line command area, order of processing etc.) that they will be treated as a group by themselves. We will therefore reserve the term *editor commands* to the group of commands referred to in the preceding paragraph. The line commands are described in detail in section "Editor Line Commands" on page 4-142.

Line commands offer a convenient way of applying multiple changes (adding, deleting, copying, moving of shifting of lines) with one interaction of the processing unit. Line commands are entered in the line command area of the first or only affected line.

Order of Input Processing

When you press ENTER or any PF key, the screen input is processed in the following sequence:

1. The data area is scanned for changed lines, which are recorded in the file as they are found.

2. The line command area is scanned for line commands (except for 'I' (Insert)). After all other line commands have been processed, the screen is scanned to process the insert commands. The second pass is only carried out if insert commands were found in the first pass.

3. The command area is inspected for commands.

4. If a PF key has been pressed, any commands associated with this key are processed. The commands are processed as part of that format area in which the cursor is found at the time the PF key is pressed. Modifications to the cursor position via the CURSOR command are also processed in this step.

Operating Modes

In *full screen edit* mode ('FS'), all editor commands and line commands are accepted and also a subset of the system commands (if entered without the '/').

Certain commands that produce more than one line of output may cause the editor to temporarily leave full screen edit mode. In such a case, the mode indicator changes to 'ED' (context edit mode) or 'LS' (list mode). While the terminal remains in either of these modes, context editor commands or system commands are not recognized; you may only press ENTER to return to full screen edit mode. See also the information under "Temporarily Leaving Full Screen Display" on page 4-39.

If you enter the INPUT or the INSERT command, full screen *editor input* mode is set. This does not change the mode indicator ('FS'), but the editor offers an empty data area for entering input data. In the line command areas, the string '*INPUT' is displayed. In editor input mode, you may still use editor and system commands as in full screen edit mode.

This is a result of the order of input processing, as described above. As soon as you press ENTER or any PF key, the editor terminates input mode and adds the entered data to the file. Then all commands found in the command areas or entered via PF key are executed in the normal sequence.

Character Translation

Normally, character data are entered from the terminal as lower case data. By default, VSE/ICCF translates the lower-case data to upper case. If, however, you had entered the 'CASE M' command, mixed-case data are stored as they were entered.

Note: *Because hexadecimal data may contain screen control characters, unprintable hexadecimal patterns are converted to X'6A' before transmission to the terminal. Thus, they are displayed as blanks or ':' and do not disturb the display format.*

String Arguments

Several of the editor commands require arguments called *string arguments.* A string argument is either matched against strings or replaces a string in the text.

A string argument may or may not begin with a delimiter character (usually a slash '/' unless changed by the DELIM command). The only commands that require the delimiter character are the CHANGE command and the string version of the DELETE command.

A string that does not begin and end with the delimiter character is assumed to begin with the first character. This includes space characters, but not the one that separates the command from the string. The string is considered to end with the last nonspace character on the line.

If a string begins or ends (but not both) with the delimiter character, the character is assumed to be part of the string.

Using Special Control Keys for Editing

The 3270 keyboard has special keys which cause certain interrupts to be passed to the editor. The following keys have a special function when used within the editor:

CLEAR Pressing the CLEAR key causes the screen to be reformatted. The screen then presents the same display as at the time when ENTER was pressed last, with the exception that the cursor will always be positioned at the beginning of the command area. Any commands entered on the screen or changes made to lines following the last ENTER will be lost.

ERASE INPUT Pressing this key has the same effect as pressing CLEAR. The only exception is that a new display does not appear unless you press the ENTER key.

ENTER Pressing the ENTER key requests the editor to read the data entered on the screen and to process any changed lines and any commands read from the screen.

FIELD MARK This key inserts a logical tab character into the data stream (unless the logical tab character has been set to some other character via the SET TAB= command).

PA1, PA2, PA3 The functions attached to these keys depend on how your system is tailored. In the VSE/ICCF system that is delivered to you, they have the following meaning:

- PA1 -- No function in VSE/ICCF but may be used as print key in CICS/VS.
- PA2 -- This key forces an editor CANCEL command to be executed. All files being edited are handled as if a QUIT command had been issued. The editor terminates. Automatic saves on newly created files will not be done; changes applied on the screen will not be processed.
- PA3 -- Same as the CLEAR key.

PF1 through 12 (for some keyboards 24). The PF keys can be set to any command, data line or multiple commands separated with logical line end characters. When a PF key is pressed, all the ENTER key functions are performed first. That is, the editor reads from the screen and processes all changed lines and commands. Then the functions associated with the PF keys are executed. For more information on PF keys and how they are used, see the descriptions of the following commands:

- /SET PFED
- /SET PFnnED
- PFnn

Entering Multiple Commands

Multiple editor commands can be typed into the command area. They must be separated by the logical line end character (see the /SET END= command in *Chapter 3. System Commands, Procedures and Macros*). As many commands as fit into the area can be specified.

Editor commands that result in leaving the current logical screen, for example: SCREEN, FILE, and QUIT, should not be followed by other commands. When leaving the full screen editor, commands that are chained to such commands will be ignored.

Repeating Commands

A command (or a series of commands separated with logical line end characters) can be repeated any number of times simply by preceding the command with an ampersand ('&'). For example, if '&FORWARD' is entered in the command area, the FORWARD command will remain on the screen allowing you to 'scroll' through the file by pressing ENTER.

Working with the Editor

Invoking the Editor

You invoke the editor through any of the three macros:

```
ED [member [passwd]]      to edit a VSE/ICCF library member , or
                          the input area
EDPRT                     to edit the print area $$PRINT
EDPUN                     to edit the punch area $$PUNCH
```

It is advisable to assign frequently used commands to PF keys. You can set and change PF keys anytime. However, the most practical way is to include such settings in a logon routine (normally a macro) which is run automatically at logon time.

To achieve this, do the following:

1. If not done already, ask your administrator to enter the name of your logon macro into your user profile (via the LOGONRTN operand of the DTSUTIL command ADD USER or ALTER USER).

2. Create your logon macro as follows:

```
ED
INPUT
@MACRO
.                                 ( other commands )
.                                 ( as required     )
/SET PF1ED  command1
/SET PF2ED  command2              ( these PF key settings will be
.                                   effective in edit mode only)
.
/SET PF12ED  command12
                                  ( null line to end input mode )
FILE name                         ( use same name as in user profile )
```

Creating a New File

A new file is created by entering the data into the input area and then saving the input area as a VSE/ICCF library member. After invoking the editor (to edit the input area, just enter 'ED' without a member name), the screen will look as in the upper part of Figure 4-4.

```
===> input_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                   /***/
***** END OF FILE *****                                   *****
```

```
===> file newdata_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                   /***/
data record 1                                             *INPUT
data record 2                                             *INPUT
data record 3                                             *INPUT
data record 4                                             *INPUT
data record 5                                             *INPUT
data record 6                                             *INPUT
data record 7                                             *INPUT
data record 8                                             *INPUT
.                                                         *INPUT
.                                                         *INPUT
.                                                         *INPUT
.                                                         *INPUT
.                                                         *INPUT
.                                                         *INPUT
.                                                         *INPUT
.                                                         *INPUT
.                                                         *INPUT
.                                                         *INPUT
last data record                                          *INPUT
                                                          *INPUT
                                                          *INPUT
```

Figure 4-4. Creating a New File

The INPUT command formats the screen for full screen data entry as shown in the second half of Figure 4-4. When entering the data records, use the NEW LINE key to start a new record. After the last record, type the FILE command as shown in the figure, and press ENTER. The editor will save the data as member NEWDATA and terminate.

All data lines are initially null lines (X'00'). When you press ENTER to terminate input mode, any remaining null lines are discarded. If you want to enter a blank line, you must type at least one blank (via the space bar) and press ENTER.

Intermediate Saving

You should take good care of saving your data, especially when entering large amounts of data. Proceed as follows:

- Issue a 'SAVE name' command after entering at least one record. This will save your data as a library member. For the rest of the session, you will be adding data to that member, not to the input area. This is a precaution against loss of your data in case of a system breakdown.

- After the initial SAVE, use '&INPUT' instead of 'INPUT'. Thus, if you press ENTER after typing one screen full of data, your data is written to the file, and the editor returns to editor input mode immediately (the data area is cleared to receive the next batch of data records).

- If all data has been entered, purge the '&INPUT' command from the command area, and press ENTER.

- If your data has been saved before, terminate the session via the QUIT command. Otherwise, use FILE as described in the previous section.

Displaying and Changing a File

Most of the data manipulation operations can be carried out in two ways: either through editor commands or through line commands. Although line commands can handle up to 999 records at a time (only 99 in case of copy or move operations), their main use lies in handling smaller portions of data which start and end on one screen.

In contrast to line commands, the editor commands start working from the current line, which often requires additional commands for exact line positioning. On the other hand, many editor commands and macros accept character string operands to denote the last record to be handled. Thus, you can avoid counting the lines down to the end of the area to be processed by a command.

Positioning the Line Pointer

You may use the following commands to position the line pointer for viewing and editing a distinctive portion of a file.

EDITOR COMMANDS	FUNCTION
Bottom	Positions the line pointer past the last line in the file.
Top	Positions the line pointer to the null line in front of the first line in the file.
FOrward	Scrolls the display forward by the specified number of pages.
BAckward	Scrolls the display back by the specified number of pages.
Next	Advances the line pointer a given number of lines.
Up	Repositions the line pointer a given number of lines before the current line.
Find	Searches the file for a string, to be found at a specific column.
Locate	Searches the file for a string, to be found at any column (unless a column is specified).
LOCUp	Searches the file backward for a string, to be found at any column (unless a column is specified).
LOCNot	Searches the file for the first non-occurrence of a given string.
ENTER key	Pressing ENTER is equivalent to entering 'NEXT 1'.

LINE COMMANDS	FUNCTION
	Positions the line pointer to the line in which the '/' had been given.

Figure 4-5. Vertical Line Positioning Commands

Associated with each format area is a so-called *line pointer*. Its setting determines which line is displayed as the *current line* (this is the first line below the scale line unless the VERIFY command specified otherwise).

The current line is the starting point for editor commands that modify, copy or move data. Only search-type commands start from the line after the current line (in case of LOCUP, it is the line before the current line). Editor commands that insert data lines, do this after the current line.

If a search-type operation finds the desired string, the line containing it becomes the current line; if the string is not found, the line pointer is set at the last record in the file. On input type operations, the last line entered becomes the current line. On line deletion, the line after the last deleted line becomes the current line.

Please note the following peculiarities:

The editor maintains a 'pseudo line' at the top of the file (***** TOP OF FILE *****). This line is displayed as the current line when the editor is invoked, or after entering the TOP command. With the line pointer at this position, you can insert data before the first record of your file. With other commands (for example DELETE), the TOP line is ignored.

Limiting Editor Operations to Certain Columns

Many applications require that certain change, shift or align operations are carried out only within a certain column range, the *editing zone*. The primary means to set this column range is the ZONE command which defines a starting and an ending column for subsequent editing operations. Data outside of that column range will be ignored by search-type commands and will not be affected by any change operation. Also, full screen changes (any new data that you typed in and any deletion of characters) outside of the editing zone are ignored once you press ENTER and will not result in a VSE/ICCF library member being changed.

In addition, a so-called *Cnn suffix* can be attached to many commands. The suffix defines column 'nn' as the starting (left) zone column. This definition is valid only for one execution of the command. The Cnn suffix may be attached to any valid command abbreviation.

The examples in the following chapters will highlight the effects of zone setting and Cnn suffix in the context of data manipulation operations.

Figure 4-6 gives an example of how to use the Cnn suffix. In this case, it is used to put the missing continuation character into column 72 of the second line of the DTFSD macro call.

```
===> next;overlayc72 X;up_
<<..+....1....+....2....+....3. .5....+.. MEM=MAINPGM >>..+..FS
FILIN    DTFSD IOAREA1=IOA,IOAREA2=IOB,IOREG=(2),       X /===/*
               TYPEFLE=INPUT,BLKSIZE=400,RECSIZE=80,      *===*
               DEVICE=3340,EOFADDR=ENDDISK                *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=MAINPGM >>..+..FS
FILIN    DTFSD IOAREA1=IOA,IOAREA2=IOB,IOREG=(2),       X /===/*
               TYPEFLE=INPUT,BLKSIZE=400,RECSIZE=80,    X *===*
               DEVICE=3340,EOFADDR=ENDDISK                *===*
```

Figure 4-6. Using the Cnn Suffix

Global Changes

You can make global, or repetitive changes, with the CHANGE and ALTER commands. In these commands, you can include operands that indicate:

- The number of lines to be searched for a character or character string. An asterisk (*) indicates that all lines, from the current line to the end of the file, are to be searched.
- Whether only the first occurrence or all occurrences on each line are to be modified. The letter 'G' indicates all occurrences. If you do not specify the letter 'G', only the first occurrence on any line is changed.

For example if you are creating a file that uses the '[' special character (X'AD') and you do not want to use the ALTER command each time you need to enter the '[' you could use the character $ as a substitute each time you need to enter a '['. When you are finished entering input, move the line pointer to the top of the file, and issue the global ALTER command:

```
alter $ ad * G
```

All occurrences of the character $ are changed to X'AD'. The line pointer is positioned at the end of the file.

When you use a global CHANGE command, you must be sure to use the final delimiter on the command line. For example,

```
change /hannible/hannibal/ 5
```

This command changes the first occurrence of the string 'HANNIBLE' on the current line and the four lines immediately following it.

You can also make global changes with the OVERLAY command, by issuing a REPEAT command just prior to the OVERLAY command. Use the REPEAT command to indicate how many lines you want to be affected. For example, if you are editing a file containing the three lines

```
A
B
C
```

with the line pointer at line 'A', issuing the commands:

```
repeat 3
overlay   I     I     I
```

results in

```
A   I     I     I
B   I     I     I
C   I     I     I
```

The line pointer is now at the line beginning with the character 'C'.

The following figure summarizes the commands that are used for making global changes.

EDITOR COMMANDS	FUNCTION
Change	Changes one string of characters to another.
Alter	Changes one character to another, or references a character by its hexadecimal value.
REPEAT	Causes the following OVERLAY or BLANK command to execute a given number of times (also used with the same effect on the commands ALIGN, CENTER, JUSTIFY, and SHIFT; see the following figure).
Overlay	Overlays a string of characters in the current line by the specified string.
BLank	Blanks out characters in the current line.
RENum	Generates new sequence numbers in the file being edited.

Figure 4-7. Global Change Commands

Shifting Data Within Lines

The following figure summarizes the commands that are used for shifting data within a line.

Figure 4-9 on page 4-16 shows how the REPEAT and CENTER commands are used in conjunction.

EDITOR COMMANDS	FUNCTION
REPEAT	Causes the following ALIGN, CENTER, JUSTIFY, or SHIFT command to execute a given number of times (also used with the same effect on the commands OVERLAY and BLANK; see the preceding figure).
ALIgn	Aligns (justifies) text to both the left and the right margin.
CENter	Centers text as determined by the current zone setting or by the column suffix.
JUStify	Aligns (justifies) text to either the left or the right margin.
SHIft	Shifts the data within the current zone left or right the specified number of columns.
LINE COMMANDS	**FUNCTION**
TA	Aligns (justifies) text to both the left and the right margin, in one or more consecutive lines.
TC	Centers text, in one or more consecutive lines.
TL	Aligns (justifies) text to the left margin, in one or more consecutive lines.
TR	Aligns (justifies) text to the right margin, in one or more consecutive lines.
>	Shifts text to the right, in one or more consecutive lines.
<	Shifts text to the left, in one or more consecutive lines.

Figure 4-8. Shift Commands

```
===> zone 1 30;repeat 2;center;top_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                    /***/
VSE/ICCF                                                   *===*
TERMINAL USER'S GUIDE                                      *===*
***** END OF FILE *****                                    *****
```

```
===> _
<<..+....1....+....2....+...>>. .5....+.. INP=*INPARA* ...+..FS
***** TOP OF FILE *****                                    /***/
           VSE/ICCF                                        *===*
    TERMINAL USER'S GUIDE                                  *===*
***** END OF FILE *****                                    *****
```

Figure 4-9. Use of REPEAT and CENTER Commands

Adding or Deleting Lines

EDITOR COMMANDS	FUNCTION
LAdd	Adds blank lines to the file.
Insert	Inserts a string as a new line in the file.
INPut	Places the terminal into editor input mode.
DELete	Deletes lines from the file.
LINE COMMANDS	**FUNCTION**
A	Adds blank lines to the file.
D	Deletes lines from the file.

Figure 4-10. Add and Delete Commands

Copying or Moving Lines

The following figure summarizes the commands that are used for copying and moving lines.

EDITOR COMMANDS	FUNCTION
DUP	Duplicates the current line the specified number of times.
@COPY	Copies lines within a file.
@MOVE	Moves lines within a file.
GETfile	Inserts data from a library member, or a work area, into the file being edited.
LINE COMMANDS	**FUNCTION**
"	Duplicates the line the specified number of times.
C	Copies lines into the editor stack. The data already in the stack is erased before the new data is stored.
K	Copies lines into the editor stack. The lines are placed behind the data that is already there.
M	Moves lines into the editor stack, deleting the lines in the file. The data already in the stack is erased before the new data is stored.
I	Inserts the lines indicated by either the 'C' command or the 'K' command after the line where 'I' is entered.

Figure 4-11. Copy and Move Commands/Macros

Using the Line Command Area

The line command areas let you easily move and copy small amounts of data from one place to another in the same file, or in different files. You can also collect lines from one or more screens (which may contain data from different files) for insertion somewhere else in the same, or in different, files.

The line commands that you need for these operations are 'M' (move), 'C' (copy), 'K' (collect or stack), and 'I' (insert).

Moving and Copying on the Same Screen

To move and copy data from one place to another on the same screen, flag each line with an 'M' or 'C' command in the line command area. If multiple contiguous lines are to be moved or copied, you need not flag each line. Instead, flag only the first and follow the 'M' or 'C' command with the number of lines to be moved or copied.

The point at which the moved or copied lines are to be inserted is indicated by placing an 'I' (insert) command in the line command area on the line after which the data is to be inserted. Data can be moved or copied upwards or downwards on the screen. Multiple 'I' commands on the screen cause the lines to be moved or copied to be inserted at each place specified.

Note that a move command deletes the data indicated by 'M' after the data was placed in the stack.

In the full screen editor, the first 'M' or 'C' command opens the stack (see STACK OPEN command). All later 'M' or 'C' commands merely add their lines to the stack. The 'I' command causes a STACK CLOSE followed by a GETFILE $$STACK command.

Moving or Copying Data From One Screen to Another

To move or copy data from one screen to another, follow the instructions in the preceding paragraphs. However, instead of placing the 'I' command in a line command area on the current screen, simply find the area (perhaps the same file or a different file) in which the data is to be inserted such that the point of insertion appears on the screen. Then just enter the 'I' command on the line after which the data is to be inserted.

Collecting Data From Several Screens for Insertion

It is also possible to collect data from several screens for insertion at a place on some other screen. To accomplish this, use the 'C' command on the first screen containing data to be copied. This will cause the stack area to be opened. Then use the 'K' command on the other screens containing data to be copied. The 'K' command will add the new data behind the existing data in the stack area, thus preventing the data from the previous screen from being overlaid.

When the point in the file is reached where the collected data is to be inserted, use the 'I' command to force the insertion.

Inserting Data via GETFILE

The GETFILE command allows you to copy all or a portion of a library member into any part of the file you are editing. For example:

```
getfile single
```

inserts all the lines of member SINGLE into your file immediately following the line pointer. The line pointer is at the last line that was read in. Or, as another example, you could specify

```
getfile double 10 25
```

to copy 25 lines, beginning with the tenth line, from the file named DOUBLE.

GETFILE can also be used to copy data from the stack, or from another part of the file you are just editing.

Another way of copying small amounts of data is to use the STACK command which is explained in the following section.

Using the Editor Stack

When you are in full screen edit mode, the punch area ($$PUNCH) associated with your terminal is used for certain editor functions. In this context, the punch area is referred to as the 'stack.' Also, the name '$$STACK' is synonymous with '$$PUNCH.'

The stack serves to temporarily store data. You store data in the stack by various means:

- The STACK command.

  This command stores in the stack any of the following:

  - Records from the member being edited.
  - Editor settings such as CASE or ZONE. The editor settings can be reset by issuing the editor command RESTORE.
  - Any commands or data specified in the STACK command.

  In the latter case, a series of commands which you wished to repeat could be placed in the stack as a temporary macro. Then specifying @$$STACK as a command name will cause the commands in the stack to be executed. The commands in the stack that contain variable parameters will be filled in from the operands of the @$$STACK macro call. (Note that the commands in the stack may also be system commands.)

  When copying **data**, you proceed as follows: Set the line pointer in the source file to the first line to be copied. Then open the stack by entering 'STACK OPEN'. Issue the 'STACK nn' command; up to 99 lines can be placed in the stack. After all lines to be copied have been stacked, close the stack with 'STACK CLOSE'. Then set the line pointer in the receiving file to the line after which the copied lines are to be inserted. Finally, enter 'GETFILE $$STACK' to copy the data to the target location.

- The CHANGE or the ALTER command.

  These commands allow, during global change operations, the storing of changed lines into the stack. After the global changes have been completed, you can issue the LIST $$STACK command to view your changes.

- Most of the commands listed in Figure 4-11 on page 4-18.

  The data to be copied or moved within a file or between files is temporarily kept in the stack area. You cannot stack more than 99 records with one 'C' or M command. However, each additional 'K' command will extend the stack by another 100 records, if required. The @COPY and @MOVE macros, on the other hand, can never handle more than 99 records.

The stacked data is preserved until new data is written into the stack. Thus, after stacking some data either via @COPY/@MOVE or 'C', 'K', 'M', or STACK commands, you may issue any number of 'I' commands to insert those data at different points of the same or another file. You may, therefore, even end the edit session and edit a different member and still refer to the previously stacked data.

Because the $$STACK area is physically the same as the $$PUNCH area, you may use the editor to view and process punch output created by the execution of a program in an interactive partition. Although you may edit (that is: change) the $$STACK area, you cannot add or delete lines in the $$STACK area.

When working in asynchronous mode, you should not use the stack because this may interfere with the execution of your program in the interactive partition. In this case, do not use any STACK or move/copy commands.

Changing and Restoring Editor Settings

The STACK EDIT and RESTORE commands are used together. The STACK EDIT command saves editor settings such as message and verification display, line number settings, tabs, control character settings, and zone setting. When you are editing a file and you want to temporarily change some of these settings, enter the STACK EDIT command to save their current status. Then, when you have finished editing, issue the RESTORE command to restore the original settings.

Note: *Make sure that a STACK EDIT command does not counteract any previous move/copy operation, or vice versa. In other words, if you had copied some data into the stack and then issued STACK EDIT, the data could be erased before you had retrieved them.*

Using Tab Stops for Column-Oriented Editing

For column-oriented editing, VSE/ICCF offers a programmed equivalent of the tabulator stops which are available on an ordinary typewriter. Using this facility is a two-step process:

1. You 'set the tab stops' at the desired columns (as you would do on a typewriter) using the editor command TABSET.

2. You define a logical TAB character and you set two PF keys. The logical TAB character is used during command and data input to represent the number of blanks needed to fill the line up to the next TAB stop. The PF keys are set equal to the editor commands 'CURSOR TABBACK' and 'CURSOR TABFORWARD' to help you position the cursor in the desired column for full screen update.

In Figure 4-12 on page 4-22 you see how the TAB characters found in the input data are expanded into the proper number of blanks when the ENTER key is pressed to end full screen data input. It is assumed that the semicolon has been defined previously as line end character.

```
===> tabset 10 16 30 72;set tab=¢;input_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                   /***/
***** END OF FILE *****                                   *****
```

```
===> top_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                   /***/
mainpgm¢csect                                            *INPUT
¢balr¢5,0¢establish addressability                       *INPUT
¢using¢*,5                                               *INPUT
¢open¢filout¢open print output                           *INPUT
loopa¢excp¢rdccb¢read card input                         *INPUT
¢wait¢rdccb                                              *INPUT
                                                         *INPUT
```

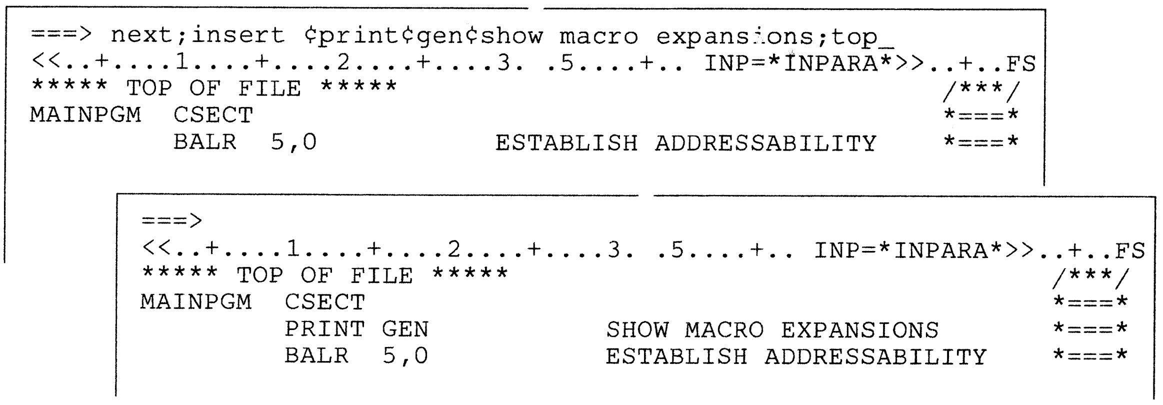
```
===> _
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                   /***/
MAINPGM  CSECT                                            *===*
         BALR  5,0            ESTABLISH ADDRESSABILITY    *===*
         USING *,5                                        *===*
         OPEN  FILOUT         OPEN PRINT OUTPUT           *===*
LOOPA    EXCP  RDCCB          READ CARD INPUT             *===*
         WAIT  EDCCB                                      *===*
***** END OF FILE *****                                   *****
```

Figure 4-12. Using TAB Stops for Column-Oriented Input

Note that the TAB characters work the same way when entered on blank lines created by the LADD or the 'A' commands.

In Figure 4-13 you see that the TAB characters may also be used within an editor command to ensure proper placement of the operand data. (TAB setting and END-character setting are the same as in Figure 4-12.)

Note that, for the purpose of interpreting TAB characters in the command line, the second byte after the command (INSERT in this case) is counted as column 1 of the data string operand.



```
===> next;insert ¢print¢gen¢show macro expansions;top_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                  /***/
MAINPGM  CSECT                                           *===*
         BALR  5,0            ESTABLISH ADDRESSABILITY    *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                  /***/
MAINPGM  CSECT                                           *===*
         PRINT GEN            SHOW MACRO EXPANSIONS       *===*
         BALR  5,0            ESTABLISH ADDRESSABILITY    *===*
```

Figure 4-13. Using TAB Stops in Editor Commands

Editing Two Areas of One File Simultaneously

When studying or changing a program, it may often occur that you want to view or edit simultaneously two different portions of a file at the same time.

The FORMAT command allows you to divide a logical screen into multiple format areas as shown in Figure 4-14. Thus, it is possible to independently edit a given file at different points. This facility is useful when trying to view one area of a file while making changes to another area, or when making matching changes to multiple areas, or when moving or copying data from one portion of a file to another.

The figure cannot show how you flip the cursor from one command area to the other. This is done most easily by setting a PF key equal to the command 'CURSOR INPUT'. This command puts the cursor into the next (or only) command area on the screen.

Assume now, you want to know, how 'RDCCB' has been defined. You enter the FORMAT command as shown, press ENTER and then the 'CURSOR INPUT' PF key. The screen is formatted as shown in part 2 of the figure and the cursor is placed into the second command area where you enter your 'find RDCCB' command. After pressing ENTER, the second format area will show the desired portion of your file (see part 3 of the figure).

If you want to return to a screen format with only one format area, just enter 'FORMAT 1'.

```
===> format 1-10 1-10_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
LOOPA    EXCP  RDCCB                                      /===/*
         WAIT  RDCCB                                      *===*
         CLC   R(3),=C'/* '                               *===*
         BE    ENDCARD                                    *===*
         MVC   0(80,2),R                                  *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
LOOPA    EXCP  RDCCB                                      /===/*
         WAIT  RDCCB                                      *===*
         CLC   R(3),=C'/* '                               *===*
         BE    ENDCARD                                    *===*
         MVC   0(80,2),R                                  *===*
         PUT   FILOUT                                     *===*
         B     LOOPA                                      *===*
ENDCARD  CLOSE FILOUT                                     *===*
         OPEN  FILIN                                      *===*
LOOPB    GET   FILIN                                      *===*
===> find rdccb_
***** TOP OF FILE *****                                   /***/
/LOAD ASSEMBLY                                            *===*
/OPTION NOSAVE                                            *===*
/FILE NAME=IJSYS01,SPACE=20,DISP=PASS                     *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
LOOPA    EXCP  RDCCB                                      /===/*
         WAIT  RDCCB                                      *===*
         CLC   R(3),=C'/* '                               *===*
         BE    ENDCARD                                    *===*
         MVC   0(80,2),R                                  *===*
         PUT   FILOUT                                     *===*
         B     LOOPA                                      *===*
ENDCARD  CLOSE FILOUT                                     *===*
         OPEN  FILIN                                      *===*
LOOPB    GET   FILIN                                      *===*
===> _
RDCCB    CCB   SYSIPT,RDCCW                               /===/*
PRCCB    CCB   SYSLST,PRCCW                               *===*
RDCCW    CCW   2,R,0,80                                   *===*
PRCCW    CCW   9,P,0,120                                  *===*
```

Figure 4-14. Using the FORMAT Command

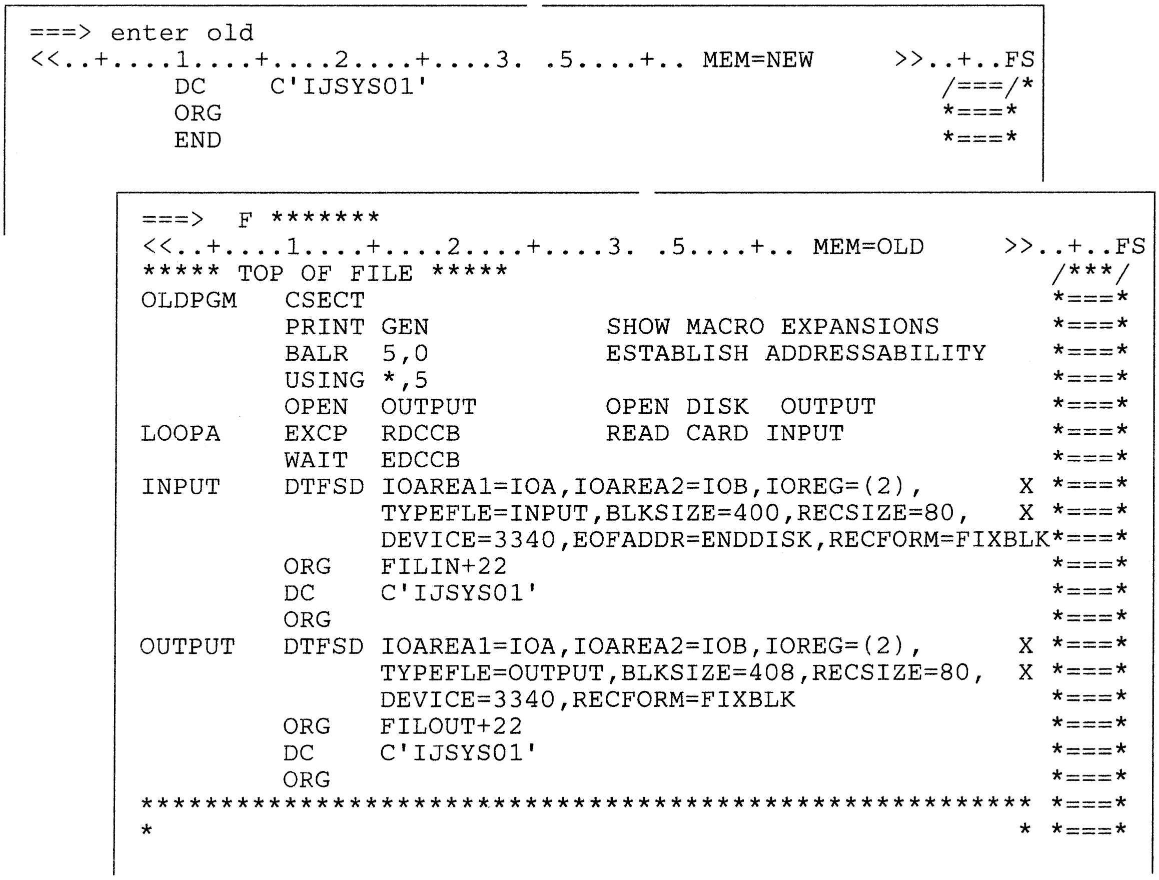
Editing Two Logically Related Files

You may experience situations where you want to copy various pieces from one file to another. The editor offers two aids: recursive editing and split screen editing; both will be described below. Recursive editing is the preferred way if larger blocks are to be copied. Split screen editing should be applied when simultaneous viewing of both files is important.

Recursive Editing

Recursive editing means alternate editing of two files within one edit session. Refer to the example in Figure 4-15 on page 4-25 and Figure 4-16 on page 4-25.

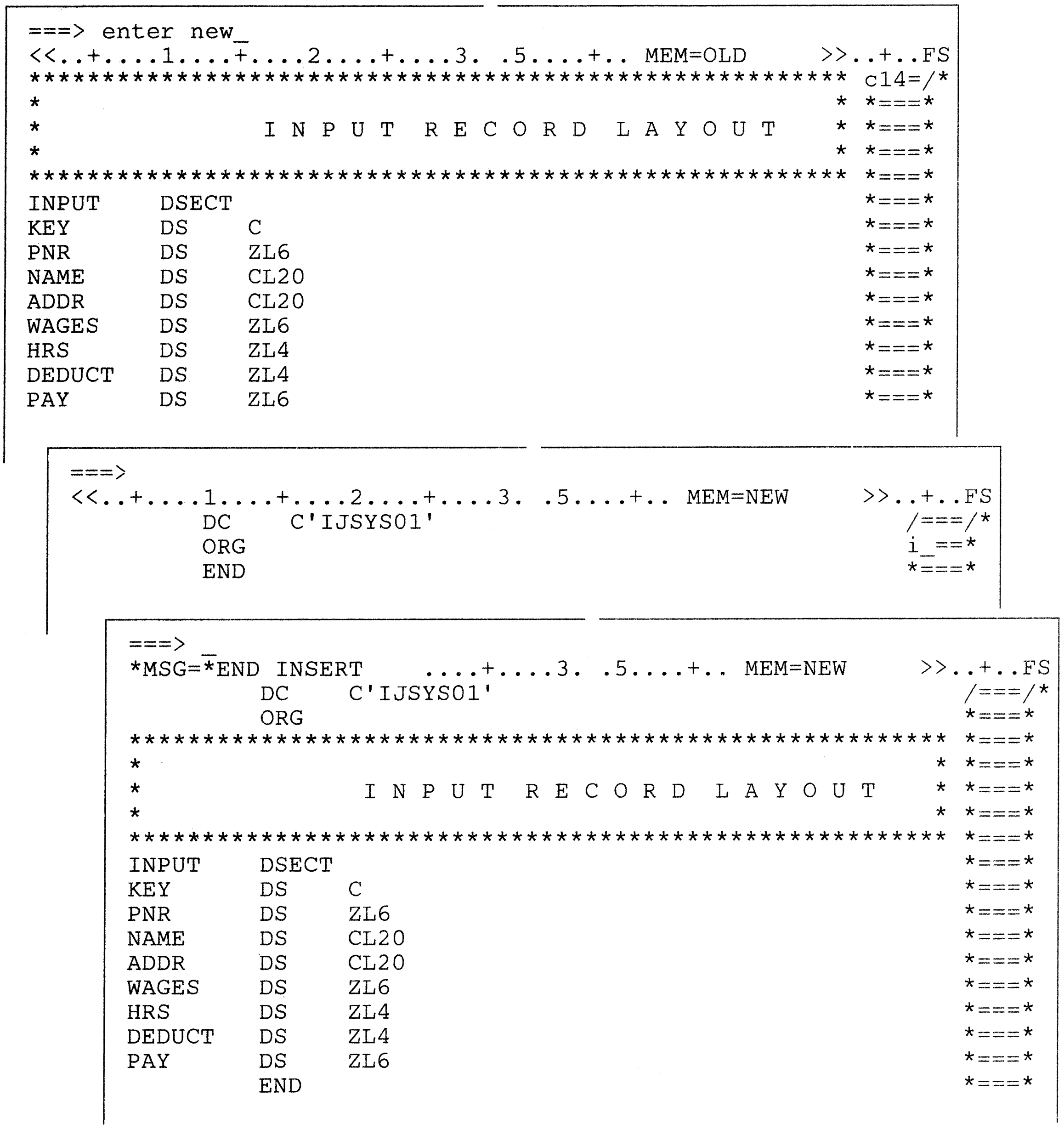
While editing member NEW, you start a new edit session for member OLD by issuing the command 'ENTER OLD'. This lets member NEW disappear from the screen, but the edit session is kept "alive", that is, the current line position and setting of edit options are maintained. With the full screen size available for the display of member OLD, it is easier to locate the data to be copied and to count the number of lines involved. After defining the data to be copied ('c14' in the line command area), an 'ENTER NEW' command brings you back to the first member. Here you insert the data from the stack ('i' in the line command area).



```
===> enter old
<<..+....1....+....2....+....3. .5....+.. MEM=NEW      >>..+..FS
         DC     C'IJSYS01'                                /===/*
         ORG                                              *===*
         END                                              *===*
```

```
===>  F *******
<<..+....1....+....2....+....3. .5....+.. MEM=OLD      >>..+..FS
***** TOP OF FILE *****                                   /***/
OLDPGM   CSECT                                            *===*
         PRINT GEN          SHOW MACRO EXPANSIONS         *===*
         BALR  5,0          ESTABLISH ADDRESSABILITY      *===*
         USING *,5                                        *===*
         OPEN  OUTPUT       OPEN DISK  OUTPUT             *===*
LOOPA    EXCP  RDCCB        READ CARD  INPUT              *===*
         WAIT  EDCCB                                      *===*
INPUT    DTFSD IOAREA1=IOA,IOAREA2=IOB,IOREG=(2),       X *===*
               TYPEFLE=INPUT,BLKSIZE=400,RECSIZE=80,    X *===*
               DEVICE=3340,EOFADDR=ENDDISK,RECFORM=FIXBLK*===*
         ORG   FILIN+22                                   *===*
         DC    C'IJSYS01'                                 *===*
         ORG                                              *===*
OUTPUT   DTFSD IOAREA1=IOA,IOAREA2=IOB,IOREG=(2),       X *===*
               TYPEFLE=OUTPUT,BLKSIZE=408,RECSIZE=80,   X *===*
               DEVICE=3340,RECFORM=FIXBLK                 *===*
         ORG   FILOUT+22                                  *===*
         DC    C'IJSYS01'                                 *===*
         ORG                                              *===*
********************************************************* *===*
*                                                       * *===*
```

Figure 4-15. Alternate Editing of Two Files (ENTER OLD)



```
===> enter new_
<<..+....1....+....2....+....3. .5....+.. MEM=OLD      >>..+..FS
****************************************************** c14=/*
*                                                    * *===*
*           I N P U T   R E C O R D   L A Y O U T    * *===*
*                                                    * *===*
****************************************************** *===*
INPUT    DSECT                                         *===*
KEY      DS    C                                       *===*
PNR      DS    ZL6                                     *===*
NAME     DS    CL20                                    *===*
ADDR     DS    CL20                                    *===*
WAGES    DS    ZL6                                     *===*
HRS      DS    ZL4                                     *===*
DEDUCT   DS    ZL4                                     *===*
PAY      DS    ZL6                                     *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=NEW      >>..+..FS
        DC     C'IJSYS01'                               /===/*
        ORG                                             i_==*
        END                                             *===*
```

```
===> _
*MSG=*END INSERT    ....+....3. .5....+.. MEM=NEW      >>..+..FS
         DC     C'IJSYS01'                              /===/*
         ORG                                            *===*
****************************************************** *===*
*                                                    * *===*
*           I N P U T   R E C O R D   L A Y O U T    * *===*
*                                                    * *===*
****************************************************** *===*
INPUT    DSECT                                         *===*
KEY      DS    C                                       *===*
PNR      DS    ZL6                                     *===*
NAME     DS    CL20                                    *===*
ADDR     DS    CL20                                    *===*
WAGES    DS    ZL6                                     *===*
HRS      DS    ZL4                                     *===*
DEDUCT   DS    ZL4                                     *===*
PAY      DS    ZL6                                     *===*
         END                                           *===*
```

Figure 4-16. Alternate Editing of Two Files (Return to NEW)

Creating a New File through Recursive Editing:

If the name specified in the ENTER command is not a member of your library, it is assumed that you want to create a new file.

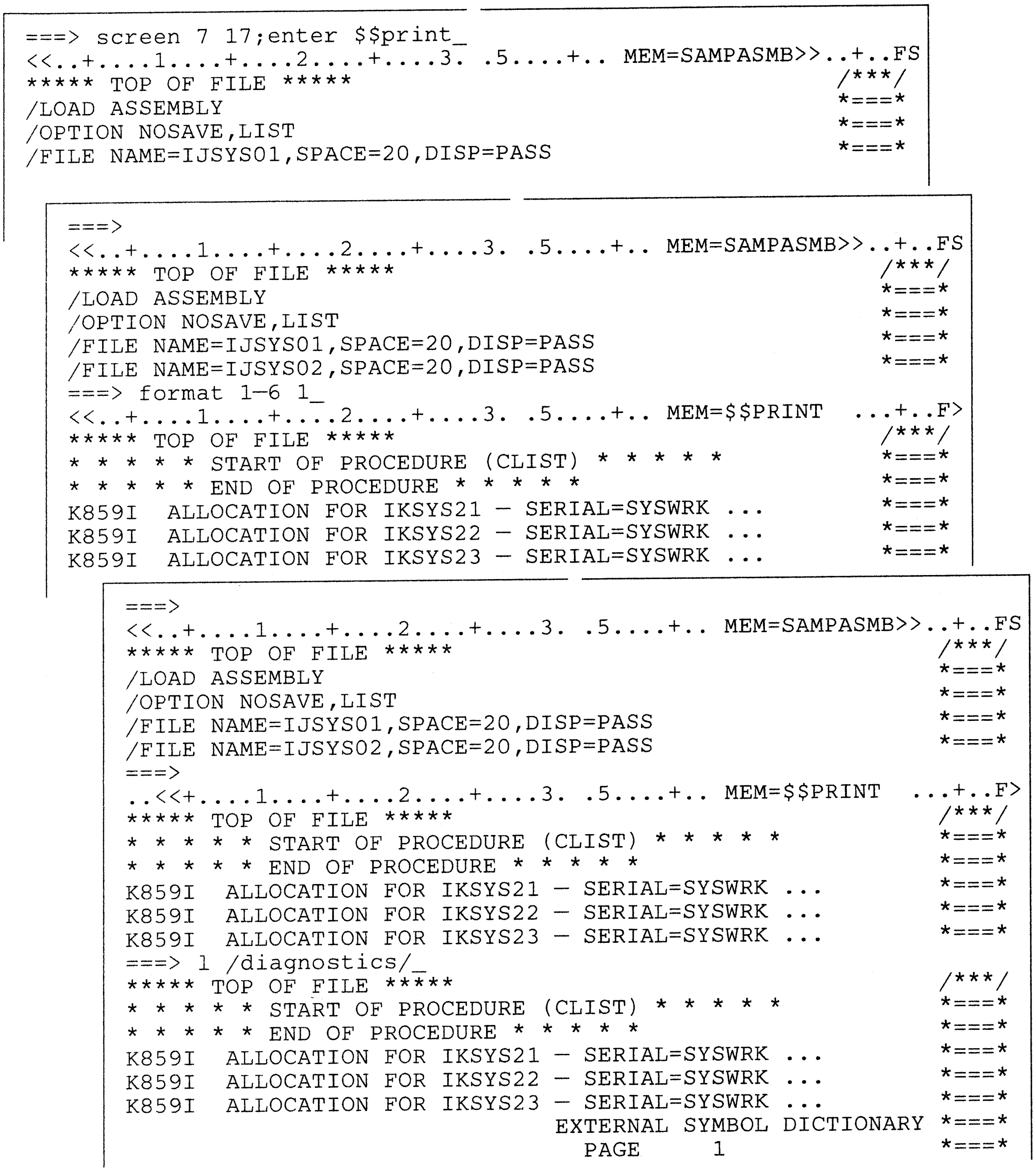
When you create a new file, you can add lines to it in a number of ways; for example with the INPUT, LADD and GETFILE commands. At this point the file is not a library member. The data that you enter is in a simulated input area (and will be lost after you entered a QUIT or CANCEL command). If you later decide to retain this data in the library as a member, you issue the SAVE or FILE command. In this way you can build several new library members without leaving edit mode.

Split Screen Editing

Split screen editing means the simultaneous editing of two files. The following example, although somewhat complicated, shows a typical application as it may occur in the process of debugging a program.

Assume you have entered an assembler program and your first assembly run shows assembly errors (with '/OPTION LIST', you get the list displayed on your screen at the end of the run). However, there are too many errors; you cannot remember all of them while browsing through the list display. So what you need is a means of looking up the diagnostics in the listing one by one while editing the program source and correcting the errors. The following two figures illustrate how to do that.

In Figure 4-17 you see how the screen is first divided into two logical screens (one for each file), and then the second logical screen divided into two format areas. (Because the number of data lines for the second format area is not specified, the editor takes whatever is available on that logical screen.) The two format areas are needed to read the error messages in the 'DIAGNOSTICS AND STATISTICS' section and to inspect the failing instructions themselves. This process is shown in Figure 4-18 on page 4-28. The third part of that figure shows that the failing instruction has been located in the program source member and may now be corrected.



```
===> screen 7 17;enter $$print_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
***** TOP OF FILE *****                                     /***/
/LOAD ASSEMBLY                                              *===*
/OPTION NOSAVE,LIST                                         *===*
/FILE NAME=IJSYS01,SPACE=20,DISP=PASS                       *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
***** TOP OF FILE *****                                     /***/
/LOAD ASSEMBLY                                              *===*
/OPTION NOSAVE,LIST                                         *===*
/FILE NAME=IJSYS01,SPACE=20,DISP=PASS                       *===*
/FILE NAME=IJSYS02,SPACE=20,DISP=PASS                       *===*
===> format 1-6 1_
<<..+....1....+....2....+....3. .5....+.. MEM=$$PRINT  ...+..F>
***** TOP OF FILE *****                                     /***/
* * * * * START OF PROCEDURE (CLIST) * * * * *               *===*
* * * * * END OF PROCEDURE * * * * *                         *===*
K859I  ALLOCATION FOR IKSYS21 - SERIAL=SYSWRK ...            *===*
K859I  ALLOCATION FOR IKSYS22 - SERIAL=SYSWRK ...            *===*
K859I  ALLOCATION FOR IKSYS23 - SERIAL=SYSWRK ...            *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
***** TOP OF FILE *****                                     /***/
/LOAD ASSEMBLY                                              *===*
/OPTION NOSAVE,LIST                                         *===*
/FILE NAME=IJSYS01,SPACE=20,DISP=PASS                       *===*
/FILE NAME=IJSYS02,SPACE=20,DISP=PASS                       *===*
===>
..<<+....1....+....2....+....3. .5....+.. MEM=$$PRINT  ...+..F>
***** TOP OF FILE *****                                     /***/
* * * * * START OF PROCEDURE (CLIST) * * * * *               *===*
* * * * * END OF PROCEDURE * * * * *                         *===*
K859I  ALLOCATION FOR IKSYS21 - SERIAL=SYSWRK ...            *===*
K859I  ALLOCATION FOR IKSYS22 - SERIAL=SYSWRK ...            *===*
K859I  ALLOCATION FOR IKSYS23 - SERIAL=SYSWRK ...            *===*
===> 1 /diagnostics/_
***** TOP OF FILE *****                                     /***/
* * * * * START OF PROCEDURE (CLIST) * * * * *               *===*
* * * * * END OF PROCEDURE * * * * *                         *===*
K859I  ALLOCATION FOR IKSYS21 - SERIAL=SYSWRK ...            *===*
K859I  ALLOCATION FOR IKSYS22 - SERIAL=SYSWRK ...            *===*
K859I  ALLOCATION FOR IKSYS23 - SERIAL=SYSWRK ...            *===*
                                  EXTERNAL SYMBOL DICTIONARY *===*
                                    PAGE     1               *===*
```

Figure 4-17. Screen Formatting for Two Files with Three Format Areas

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
***** TOP OF FILE *****                                   /***/
/LOAD ASSEMBLY                                            *===*
/OPTION NOSAVE,LIST                                       *===*
/FILE NAME=IJSYS01,SPACE=20,DISP=PASS                     *===*
/FILE NAME=IJSYS02,SPACE=20,DISP=PASS                     *===*
===> 1 / 25/_
..<<+....1....+....2....+....3. .5....+.. MEM=$$PRINT   ...+..F>
***** TOP OF FILE *****                                   /***/
* * * * * START OF PROCEDURE (CLIST) * * * * *            *===*
* * * * * END OF PROCEDURE * * * * *                      *===*
K859I  ALLOCATION FOR IKSYS21 - SERIAL=SYSWRK ...         *===*
K859I  ALLOCATION FOR IKSYS22 - SERIAL=SYSWRK ...         *===*
K859I  ALLOCATION FOR IKSYS23 - SERIAL=SYSWRK ...         *===*
===>
                              DIAGNOSTICS AND STATISTICS /===/*
                                PAGE      9               *===*
  STMNT ERROR NO.  MESSAGE                                *===*
                                   84-04-10               *===*
    25   IPK163    ADDRESSABILITY ERROR IN OPERAND 2      *===*
    27   IPK156    SYMBOL 'R2' UNDEFINED                  *===*
```

```
===> L /(3),C/_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
***** TOP OF FILE *****                                   /***/
/LOAD ASSEMBLY                                            *===*
/OPTION NOSAVE,LIST                                       *===*
/FILE NAME=IJSYS01,SPACE=20,DISP=PASS                     *===*
/FILE NAME=IJSYS02,SPACE=20,DISP=PASS                     *===*
===>
..<<+....1....+....2....+....3. .5....+.. MEM=$$PRINT   ...+..F>
000026 0000 0000 0000          25     CLC   R(3),C'/* ' /===/*
         *** ERROR ***                                    *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
         CLC    R(3),C'/* '     LAST CARD?                /===/*
         BE     ENDCARD         EXIT LOOP IF YES          *===*
         MVC    0(80,R2),R      CARD- TO DISK BUFFER      *===*
         PUT    FILOUT          WRITE TO DISK             *===*
         B      LOPA            LOOP UNTIL "/*" CARD      *===*
===>
..<<+....1....+....2....+....3. .5....+.. MEM=$$PRINT   ...+..F>
000026 0000 0000 0000          25     CLC   R(3),C'/* ' /===/*
         *** ERROR ***                                    *===*
                                                          *===*
```

Figure 4-18. Simultaneous Editing of Two Files Using Three Format Areas

Saving Your Data and Terminating the Editor

You can save your data and terminate an edit session in various ways.

- PA2 key or CANCEL command: terminates the edit session regardless of the number of editing levels being active. Data created as a 'NEW member' (that is, via the ENTER command) is lost. (Remember that as Cancel key a PA key different from PA2 may have been selected by the VSE/ICCF administrator.)

- QUIT or END: these commands return to the next higher edit level. Data created as a 'NEW member' is lost.

- SAVE or REPLACE: these commands store the new or changed data into a VSE/ICCF library member. REPLACE places the data into an existing member whereas SAVE creates a new member.

  The editor remains at the current edit level.

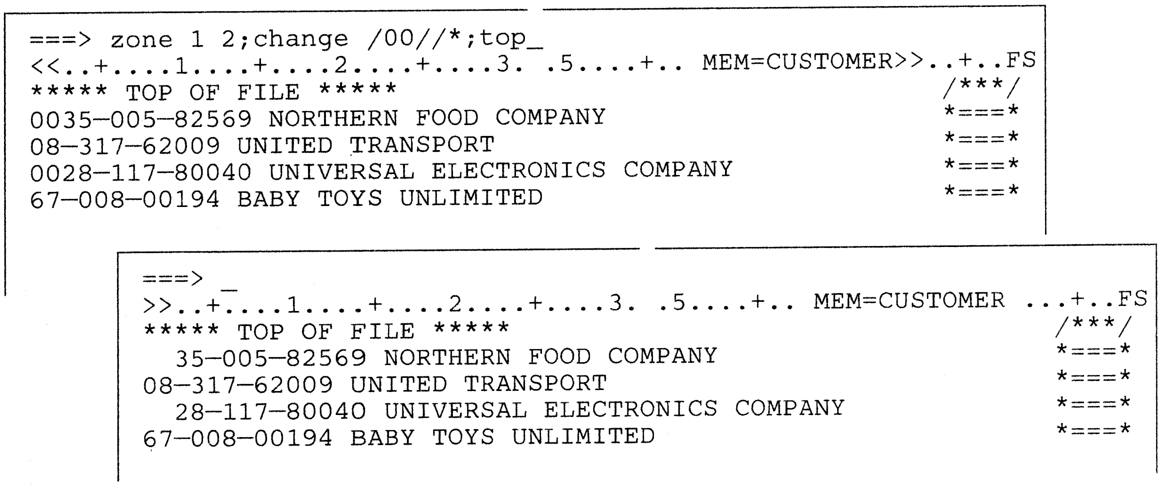
- FILE: this command performs a SAVE followed by a QUIT.

Special Applications and Techniques

Difficult Global Changes

In the following, it will be demonstrated how the ZONE command can be used to perform difficult global changes.

The file shown in Figure 4-19 contains records of similar format, but some of them have leading zeros in the district code, others do not. Assume that the leading zeros are to be removed such that all the records appear properly aligned. Because pairs of zeros are also found at other places in the file, you must use the ZONE command to limit the change to the first two columns.

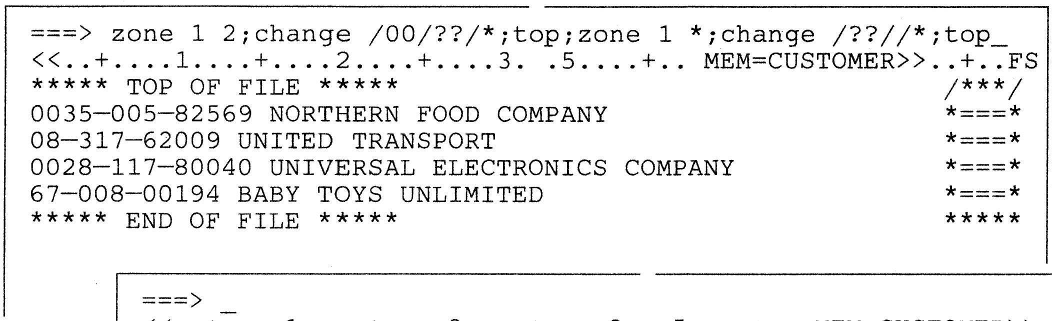


```
===> zone 1 2;change /00//*;top_
<<..+....1....+....2....+....3. .5....+.. MEM=CUSTOMER>>..+..FS
***** TOP OF FILE *****                                   /***/
0035–005–82569 NORTHERN FOOD COMPANY                      *===*
08–317–62009 UNITED TRANSPORT                             *===*
0028–117–80040 UNIVERSAL ELECTRONICS COMPANY              *===*
67–008–00194 BABY TOYS UNLIMITED                          *===*
```

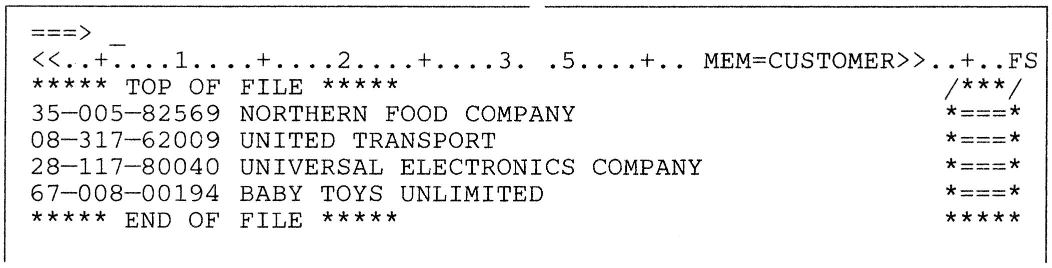
```
===> _
>>..+....1....+....2....+....3. .5....+.. MEM=CUSTOMER ...+..FS
***** TOP OF FILE *****                                   /***/
  35–005–82569 NORTHERN FOOD COMPANY                      *===*
08–317–62009 UNITED TRANSPORT                             *===*
  28–117–80040 UNIVERSAL ELECTRONICS COMPANY              *===*
67–008–00194 BABY TOYS UNLIMITED                          *===*
```

Figure 4-19. Incorrect Application of the ZONE Command

The second part of Figure 4-19 shows that a simple change command does not achieve the desired result because the zone setting prevents the rest of the record from being shifted left. Figure 4-20 indicates how you can bypass that problem.



```
===> zone 1 2;change /00/??/*;top;zone 1 *;change /??//*;top_
<<..+....1....+....2....+....3. .5....+.. MEM=CUSTOMER>>..+..FS
***** TOP OF FILE *****                                    /***/
0035-005-82569 NORTHERN FOOD COMPANY                       *===*
08-317-62009 UNITED TRANSPORT                              *===*
0028-117-80040 UNIVERSAL ELECTRONICS COMPANY               *===*
67-008-00194 BABY TOYS UNLIMITED                           *===*
***** END OF FILE *****                                    *****
```



```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=CUSTOMER>>..+..FS
***** TOP OF FILE *****                                    /***/
35-005-82569 NORTHERN FOOD COMPANY                         *===*
08-317-62009 UNITED TRANSPORT                              *===*
28-117-80040 UNIVERSAL ELECTRONICS COMPANY                 *===*
67-008-00194 BABY TOYS UNLIMITED                           *===*
***** END OF FILE *****                                    *****
```

Figure 4-20. Use of the ZONE Command for Complex Global Changes

The first change only converts the zeros into some other unique characters. After that, you do not need the zone restriction any more. You use a second change command to eliminate those characters.

Linenumber Editing

After a file has grown to a certain size, locating a particular record may become difficult; similar records may exist elsewhere in the file, or you just cannot remember whether the desired record is up or down relative to the current line. Because LOCATE operations are relatively slow on large members, you should use, where possible, linenumber editing.

Linenumber editing means that sequence numbers found in certain columns of your file are used to locate a particular record. You enter the desired line number, and the editor will search the file (in forward direction) for that line number.

Linenumber editing can be of advantage when you know the number of the line you are looking for, for example, when you have a listing with line numbers available.

If the desired line number is smaller than the current one, the search will begin at the top of the file. If your file is very large and the line to be searched for is only a few lines above the current line, it may be quicker to use the UP, BACKWARD, or LOCUP command to go backward in the file.

Figure 4-21 on page 4-32 shows you how to get started with linenumber editing. If you work as suggested in the figure, the LINEMODE command starts automatic generation of linenumbers during data input. The PROMPT command can be used to set the linenumber increment to any value. Later, when applying changes to the member, the LINEMODE command tells the editor in which columns to find the linenumber if a line is to be located by number. For the most frequently used settings, you need not specify the exact column numbers: 'LINEMODE RIGHT' automatically selects columns 73 through 80 (for assembler programs and the like), while 'LINEMODE LEFT' selects columns 1 through 5 (for VS BASIC programs).

Note: *When entering data with 'LINEMODE LEFT', you must use tabsetting and logical tab characters (see "Using Tab Stops for Column-Oriented Editing" on page 4-21) to get your data moved to column 6 (otherwise, the generated line numbers will overlay the first five columns of your data). To facilitate full-screen update, you should also set two PF keys equal to 'CURSOR TABFORWARD' and 'CURSOR TABBACK' commands.*

If you want to add line numbers to an existing member, you issue the LINEMODE command, followed by a RENUM command. By default, the RENUM command creates or updates the line numbers within the columns defined by the LINEMODE command.

When you work with 'LINEMODE RIGHT', the line command area would hide your line numbers. To avoid this inconvenience, issue the 'SET NUMBERS ON', and you get columns 75 through 80 of the line numbers displayed in the line command areas.

Note: *With 'NUMBERS ON', you can still use line commands; however, they must be entered starting in column 1 of the line command area, and they must be followed with a blank.*

Setting NUMBERS ON can be utilized with COBOL programs, too. Entering the commands

```
linemode 1 6              (establish linenumber editing)
view 7 80                 (display columns 7-80 of the file beginning
set numbers on                                in columns of the screen)
```

causes the body of the COBOL statement to be displayed at the left of the screen. You can then modify lines without having to move the cursor across the sequence number. Also, if linenumber editing is in effect, 'SET NUMBERS ON' causes the sequence number from the line number columns (rather than from columns 73-80) to be displayed in the line command area.

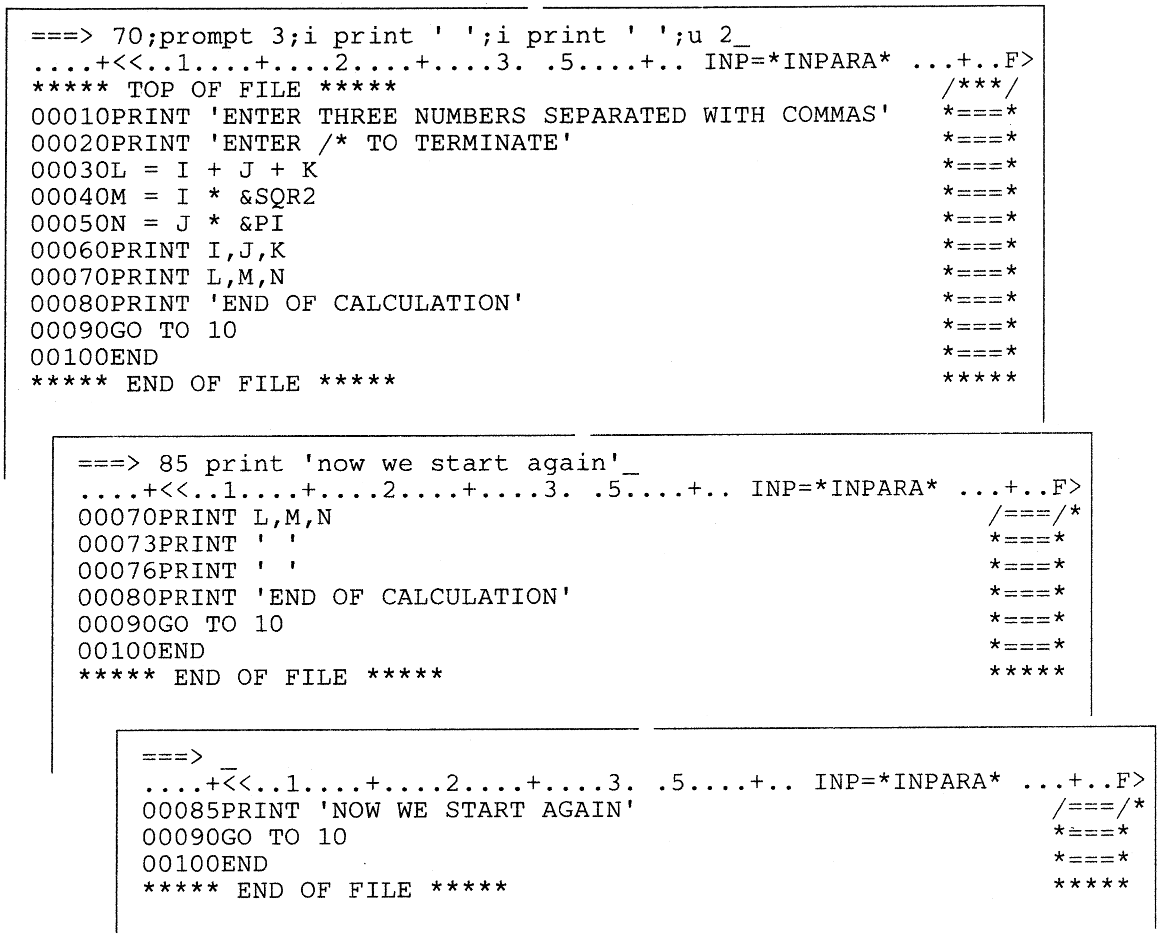
```
===> tabset 1 6;set tab=¢;linemode left;input_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                    /***/
***** END OF FILE *****                                    *****
```

```
===> top_
....+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
***** TOP OF FILE *****                                    /***/
¢print 'enter three numbers separated with commas'        *INPUT
¢print 'enter /* to terminate'                            *INPUT
¢l = i + j + k                                            *INPUT
¢m = I * &sqr2                                            *INPUT
¢n = j * &pi                                              *INPUT
¢print i,j,k                                              *INPUT
¢print l,m,n                                              *INPUT
¢print 'end of calculation'                               *INPUT
¢go to 10                                                 *INPUT
¢end                                                      *INPUT
```

```
===> _
....+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
***** TOP OF FILE *****                                    /***/
00010PRINT 'ENTER THREE NUMBERS SEPARATED WITH COMMAS'     *===*
00020PRINT 'ENTER /* TO TERMINATE'                         *===*
00030L = I + J + K                                         *===*
00040M = I * &SQR2                                         *===*
00050N = J * &PI                                           *===*
00060PRINT I,J,K                                           *===*
00070PRINT L,M,N                                           *===*
00080PRINT 'END OF CALCULATION'                            *===*
00090GO TO 10                                              *===*
00100END                                                   *===*
***** END OF FILE *****                                    *****
```

Figure 4-21. Linenumber Editing (Input)

The following figure illustrates the use of the editor command 'nn'. In the first part, line number 70 is to be located. The second screen shows how the 'nn' command is used to add a line. The figure also shows an application of the PROMPT command: the numbers for the two inserted lines (I PRINT ' ') will be built with increments of 3.



```
===> 70;prompt 3;i print ' ';i print ' ';u 2_
....+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
***** TOP OF FILE *****                                    /***/
00010PRINT 'ENTER THREE NUMBERS SEPARATED WITH COMMAS'     *===*
00020PRINT 'ENTER /* TO TERMINATE'                         *===*
00030L = I + J + K                                         *===*
00040M = I * &SQR2                                         *===*
00050N = J * &PI                                           *===*
00060PRINT I,J,K                                           *===*
00070PRINT L,M,N                                           *===*
00080PRINT 'END OF CALCULATION'                            *===*
00090GO TO 10                                              *===*
00100END                                                   *===*
***** END OF FILE *****                                    *****
```

```
===> 85 print 'now we start again'_
....+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
00070PRINT L,M,N                                           /===/*
00073PRINT ' '                                             *===*
00076PRINT ' '                                             *===*
00080PRINT 'END OF CALCULATION'                            *===*
00090GO TO 10                                              *===*
00100END                                                   *===*
***** END OF FILE *****                                    *****
```

```
===> _
....+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
00085PRINT 'NOW WE START AGAIN'                            /===/*
00090GO TO 10                                              *===*
00100END                                                   *===*
***** END OF FILE *****                                    *****
```

Figure 4-22. Linenumber Editing (Extension)

Flagging Changes

Via the FLAG editor command or the /PROTECT system command, you can request that all editor functions which change the data within a member will cause the changes to be noted ('flagged') in the changed lines themselves. For example, you may want changes to be noted in columns 73-80. The change flag contains the following information:

```
Type of change (position 1)
Date of change (positions 2-4)
Userid         (positions 5-8) of the user who
                requested the change
```

Through the VSE/ICCF tailoring option EDFLAG, the VSE/ICCF administrator can choose which columns are to contain the change flag.

Editing All 80 Columns

Normally, columns 73-80 of the file being edited are "hidden" behind the line command area. The editor offers the following ways to use these columns for data (you must set the editing zone to 1-80):

1. Using the LEFT and RIGHT commands to shift the display of your data in the indicated direction.

This method works fine, particularly, if you assign the LEFT and RIGHT commands to PF keys. However, you never see all 80 columns at the same time.

2. Using the VIEW command to display only the desired column range(s).

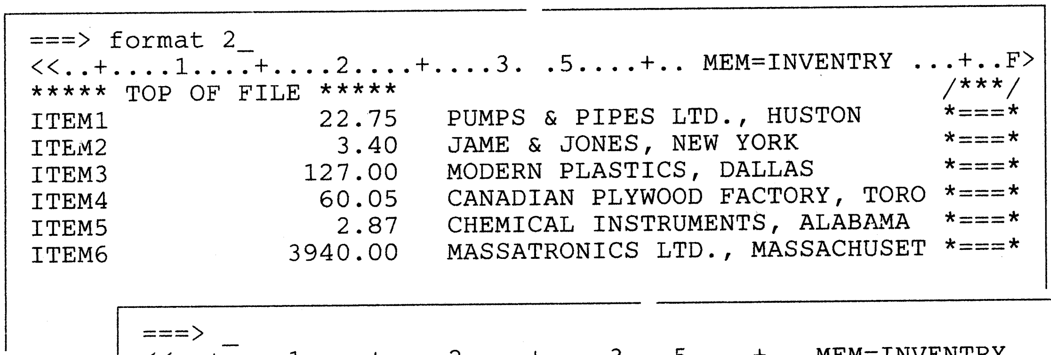
   This method offers an advantage over the first one if there is, somewhere in the middle of your file, a range of at least eight columns that needs not be displayed or changed.

3. Using the FORMAT command to assign two physical screen lines to each data record (see Figure 4-23).

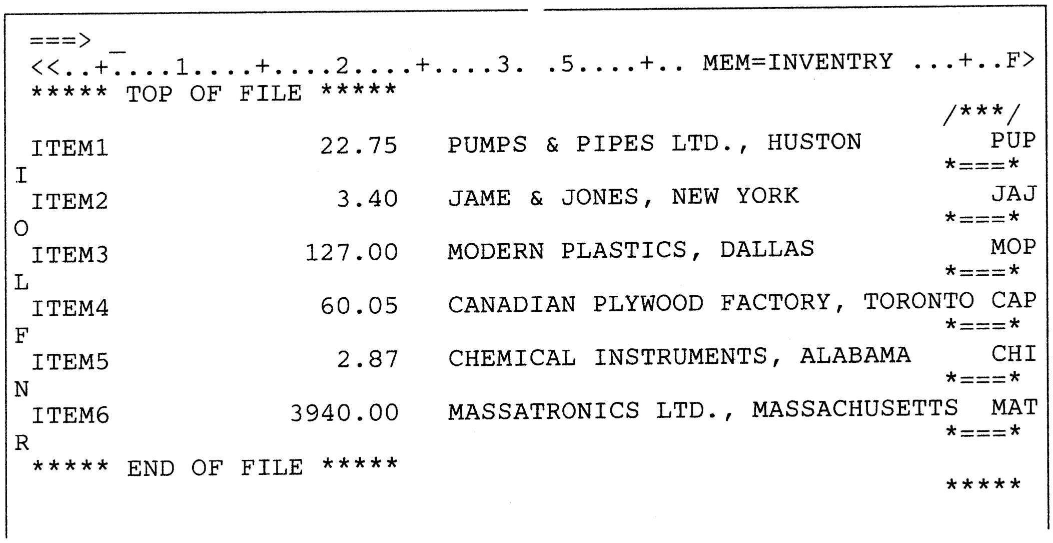
   This method allows simultaneous viewing and editing of all 80 columns; however, the double-line display may appear somewhat confusing.

4. Using the FORMAT command to eliminate the line command areas (see Figure 4-24).

   This method produces the clearest display, but you cannot use line commands any more, and column 80 stays invisible (in many cases, the file can be reorganized to leave column 80 unused).

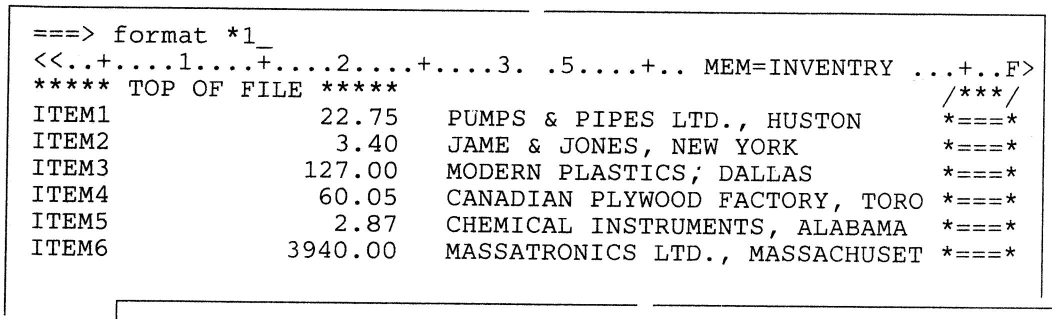


```
===> format 2_
<<..+....1....+....2....+....3. .5....+.. MEM=INVENTRY ...+..F>
***** TOP OF FILE *****                                 /***/
ITEM1              22.75   PUMPS & PIPES LTD., HUSTON      *===*
ITEM2               3.40   JAME & JONES, NEW YORK          *===*
ITEM3             127.00   MODERN PLASTICS, DALLAS         *===*
ITEM4              60.05   CANADIAN PLYWOOD FACTORY, TORO  *===*
ITEM5               2.87   CHEMICAL INSTRUMENTS, ALABAMA   *===*
ITEM6            3940.00   MASSATRONICS LTD., MASSACHUSET  *===*
```

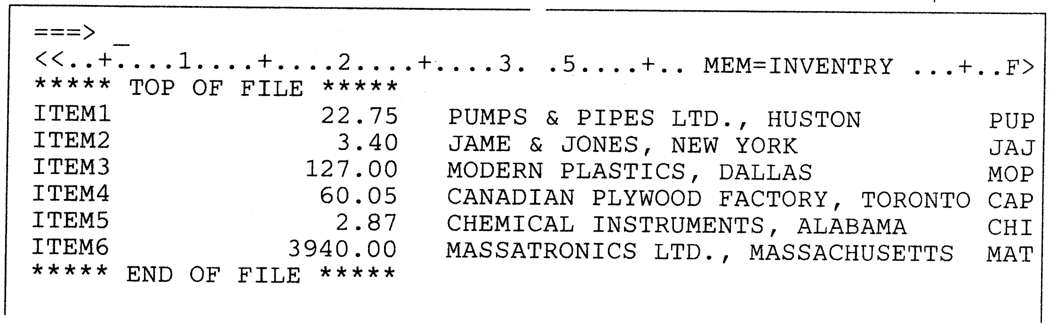


```
 ===> _
 <<..+....1....+....2....+....3. .5....+.. MEM=INVENTRY ...+..F>
 ***** TOP OF FILE *****
                                                             /***/
 ITEM1              22.75   PUMPS & PIPES LTD., HUSTON         PUP
I                                                            *===*
 ITEM2               3.40   JAME & JONES, NEW YORK             JAJ
O                                                            *===*
 ITEM3             127.00   MODERN PLASTICS, DALLAS            MOP
L                                                            *===*
 ITEM4              60.05   CANADIAN PLYWOOD FACTORY, TORONTO CAP
F                                                            *===*
 ITEM5               2.87   CHEMICAL INSTRUMENTS, ALABAMA      CHI
N                                                            *===*
 ITEM6            3940.00   MASSATRONICS LTD., MASSACHUSETTS  MAT
R                                                            *===*
 ***** END OF FILE *****
                                                             *****
```

Figure 4-23. Screen Format Using Two Lines for Each Record



```
===> format *1_
<<..+....1....+....2....+....3. .5....+.. MEM=INVENTRY ...+..F>
***** TOP OF FILE *****                                   /***/
ITEM1              22.75   PUMPS & PIPES LTD., HUSTON     *===*
ITEM2               3.40   JAME & JONES, NEW YORK         *===*
ITEM3             127.00   MODERN PLASTICS, DALLAS        *===*
ITEM4              60.05   CANADIAN PLYWOOD FACTORY, TORO *===*
ITEM5               2.87   CHEMICAL INSTRUMENTS, ALABAMA  *===*
ITEM6            3940.00   MASSATRONICS LTD., MASSACHUSET *===*
```



```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=INVENTRY ...+..F>
***** TOP OF FILE *****
ITEM1              22.75   PUMPS & PIPES LTD., HUSTON          PUP
ITEM2               3.40   JAME & JONES, NEW YORK              JAJ
ITEM3             127.00   MODERN PLASTICS, DALLAS             MOP
ITEM4              60.05   CANADIAN PLYWOOD FACTORY, TORONTO   CAP
ITEM5               2.87   CHEMICAL INSTRUMENTS, ALABAMA       CHI
ITEM6            3940.00   MASSATRONICS LTD., MASSACHUSETTS    MAT
***** END OF FILE *****
```

Figure 4-24. Screen Format for Viewing 79 Columns

Filetype Dependent Setting of Editor Options

If you work with different data formats such as assembler programs, PL/I programs, text files etc., you may find it convenient to have certain edit controls set automatically to fit the respective data format. This is done most easily by creating copies of the ED macro under new names and adding the required commands at the end of these copies.

The following example illustrates how to create a macro that makes the editing of PL/I programs easier:

You enter (as administrator) the following commands:

```
/sw 2                          (switch to common library, assuming
                                library 2 is your common library)
ed                             (edit the input area)
getfile ed                     (line pointer will be at end-of-file)
input                          (add statements below)
set tab=#                      (select character not used in PL/I)
tabset pli                     (set tabs suitable for PL/I)
set pf10ed cursor tabback      (for easy cursor positioning
set pf11ed cursor tabforward    during full screen changes)
echo tab character = number sign (remind user of tab character setting)
                               (null line to end input mode)
file edp                       (create member EDP)
```

From now on, you may use 'EDP membername' for editing PL/I programs, and you need no extra work to prepare for tabsetting and cursor positioning.

Indexed Editing for Large Members

The editor offers two aids to make editing of large members easier and quicker:

- Linenumber editing.
- Indexed editing.

The advantages and techniques of linenumber editing are described in "Linenumber Editing" on page 4-31.

In indexed editing, you use the INDEX command to create an internal index. The entries of this index point directly to every n-th record in the member ('n' being the index interval specified with the INDEX command). Then, issue the POINT command to set the line pointer at the desired record. The POINT command uses the index information to avoid sequential searches through the file wherever possible.

If the LINEMODE command is issued before the INDEX command, the line numbers of the selected lines are stored in the index, too. Thus, also the 'nn' command (locate / add / replace by line number) would utilize the index.

Note that the index is not updated in case of record additions or deletions. The 'POINT nn' command which should point to the nn-th record in a member, may become somewhat "unprecise" after record additions or deletions. Therefore, after a certain number of additions or deletions, the INDEX command should be reissued.

Indexed editing is recommended with members larger than 500 records. The index interval should be 1/30 of the member size, but must lie between 40 and 400. You may use the /COUNT command to find out the member size.

The index can cover a maximum of 12000 records. When editing larger members, the performance gain through indexing decreases when you edit beyond that limit.

Special Views of Data

When editing column oriented data (for example an RPG program), you may use the VIEW command to rearrange the way in which the columns are displayed. Up to 12 fields (column ranges) may be selected from the file and displayed in any order and any position on the screen.

Hexadecimal Editing

In certain situations, you may want to edit nonprintable hexadecimal patterns (for example apply a quick patch to an object deck). Again, the VIEW command allows to display any field in your records in two-digit hexadecimal format. Changes must be entered as hexadecimal digits. For a detailed example, refer to the description of the VIEW command.

Editing Mixed Data

If you work at an IBM 5550 with 3270 Emulation, you can edit mixed data other than DBCS print-type members with the Full Screen Editor. In Figure 4-25 on page 4-40 the editor commands which support mixed data editing are marked with a 'Y' in the 'DBCS SUPPORT' column. The commands marked with 'N' in this column do not apply to or do not support the editing of mixed data.

Maintaining Multiple Change Levels of a Member

When developing a program or a document, you may feel the need to retain older versions of it for various purposes. VSE/ICCF supports this in a convenient way through so-called *generation member groups.*

A generation member group comprises from 2 to 10 versions of a member. The member names consist of a *root* of up to six characters plus a suffix of the form '-n'. The 'n' is a decimal digit (0-9) denoting the change level (0 is the most recent update). In addition, VSE/ICCF maintains, in the directory entries of these members, flags which identify them as part of a generation member group.

Whenever a new update of the member is added to the group with a suffix of '-0', the suffixes of all other members in the group are increased by one. If all positions in the group are occupied, the oldest version of the member is purged. A new update level is added when you save the contents of the input area as '-0' member of the group (see examples below). Only level '-0' of a generation member group may be edited, older levels cannot be changed unless you 'ungroup' the members again.

The following example for creation and usage of a generation member group is based on the assumption that not each minimal change is put into the group but only selected versions of the member. Therefore, a working copy is maintained which replaces the level '-0' member of the group when certain checkpoints in the development process have been reached:

```
/group create myprog 5          (creates dummy members MYPROG-0,
                                 MYPROG-1, .... MYPROG-4)
ed myprog                       (this optional working copy is not part
    .                            of the group; it is used to accumulate
    .                            changes up to certain checkpoints.)
    .
quit                            (after applying all changes and after
                                 sufficient testing, you may put the
                                 member into the group.)

/input                          (these commands will copy the current
/insert myprog                   change level of MYPROG as member '-0'
/save myprog-0                   of the MYPROG group.)
```

The following example works without a working member; each change is put into the group.

```
/group create myprog 5          (creates dummy members MYPROG-0,
                                 MYPROG-1, ... MYPROG-4.)
ed
get myprog-0                    (all changes will be based on the latest
    .                            version in the generation member group.)
    .
file myprog-0                   (files new version of MYPROG as member
                                 '-0' of the generation member group.)
```

Logging

The VSE/ICCF logging option (set via 'SET LOG ON') is valid for full screen editing operations. Note, however, that what you see in the log file are sometimes the 'equivalents' to the data manipulation functions that you perform on the screen. For example, if lines on the screen are changed, the changed lines are internally passed as replace commands (hence the RC01 in the log area). Similarly, other screen and line command functions will appear as equivalent editor commands. Most editor commands are logged as they are processed. (A few editor command functions are not logged at all.)

Temporarily Leaving Full Screen Display

Certain editor commands such as PRINT, PRINTFWD, LIBRARY, SHOW, and HARDCPY cause your terminal to temporarily leave full screen mode. The output from these commands is not displayed in full screen editing format. To reinstate the full screen edit display after a SHOW or HARDCPY command, press ENTER or the CLEAR key. If the LIBRARY, PRINT or PRINTFWD command displays a single screen or less and returns to edit mode (indicated by 'ED' in the scale line), the full screen editor formatted display can be reinstated by pressing the ENTER or the CLEAR key.

If the LIBRARY, PRINT, or PRINTFWD commands cause more than a single screen to be displayed, your terminal will be in list mode (as indicated by 'LS' in the scale line) until the final page is reached. Succeeding pages can be displayed by pressing ENTER. While in list mode, system commands (such as /SKIP, /CONTINU, /CANCEL, etc.) will be effective. If you want to terminate list mode before the final page is reached, enter the /CANCEL command or press the Cancel key. At this point, the *END PRINT message should appear, and your terminal will reenter the full screen editor. Now press CLEAR or ENTER to reinstate the full screen editor formatted display.

Because the PRINT, PRINTFWD, LIBRARY, SHOW, and HARDCPY commands cause the terminal to temporarily leave the full screen editor, no editor command should be entered on the same screen following one of these commands or it will be ignored. Thus, for example, if a PRINT command is entered followed by a logical line end character and one or more editor commands, the commands following the PRINT command will not be executed. This also applies if there are multiple command areas on the screen. If a PRINT or SHOW command, for example, is entered in one command area, any commands in command areas farther down the screen will be ignored.

Summary of Editor Commands and Macros

COMMAND	FUNCTION	DBCS SUPPORT
Add	Adds data beyond the end of data within the edit zone.	N
ALIgn	Aligns data in the zone to right and left margins.	N
ALter	Changes a single character to another character in one or more lines.	N
BAckward	Pages backward through the file.	Y
BLank	Blanks out characters in the current line.	N
Bottom	Moves the line pointer past end—of—file.	Y
CANcel	Terminates the editor.	Y
CAse	Controls upper/lower case input data translation.	N
CENter	Centers the data within the current zone.	N
Change	Changes one character string to another in one or more lines.	Y
@COPY	(an editor macro) Copies lines from one part of a file to another.	Y
CTL	See [/]SET command.	Y
CURsor	Positions the cursor on the screen.	Y
DELete	Deletes one or more lines.	Y
DELIM	Defines the delimiter character.	Y
DOwn	See NEXT command.	Y
DUP	Duplicates the current line a specified number of times.	Y

Figure 4-25 (Part 1 of 5). Summary of Editor Commands and Macros

COMMAND	FUNCTION	DBCS SUPPORT
ECHO	See [/]ECHO system command.	Y
END	See QUIT command.	Y
ENTer	Starts editing an additional file.	Y
FILe	Saves the input area file or a newly created file in the library and terminates editing of that file.	Y
Find	Performs a column dependent search for specified data.	Y
FLag	Sets change flagging on or off.	Y
FORMat	Defines the number and configuration of format areas within a logical screen.	Y
FOrward	Pages forward through the file.	Y
@FSEDPF	(an editor macro) Sets PF keys for use in full screen edit mode.	Y
GETfile	Copies data from another member or work area into the file that you are editing.	Y
HARdcpy	See [/]HARDCPY system command.	N
INDex	Builds an index of the current file for rapid editing.	Y
INPut	Places a format area into editor input mode.	Y
Insert	Inserts a new line into the file.	Y
JUStify	Left or right justifies data within the zone.	N
LAdd	Adds blank lines to the file.	Y
LEft	Shifts (the 'view' is shifted right) the data within the logical screen to the left.	Y
LIBRARY	See [/]LIBRARY system command.	Y

Figure 4-25 (Part 2 of 5). Summary of Editor Commands and Macros

COMMAND	FUNCTION	DBCS SUPPORT
LINemode	Sets line number editing on or off.	Y
LN	See LOCATE command.	Y
Locate	Locates a string of characters within a file.	Y
LOCNot	Locates the first nonoccurrence of a string.	Y
LOCUp	See LOCATE command.	Y
LUp	See LOCATE command.	Y
MSG	See [/]MSG system command.	Y
@MOVE	(an editor macro) Moves lines from one part of a file to another; see @COPY editor macro.	Y
Next	Moves the line pointer forward a given number of lines.	Y
Overlay	Overlays the current line with the characters specified in the operand field of the command.	N
OVERLAYX OX	Overlays the current line with hexadecimal format data.	N
PF	See PRINT command.	Y
PFnn	Simulates the function of a PF key.	Y
POint	Positions the line pointer based on an established index.	Y
Print	Displays a given number of lines.	Y
PRINTFwd	See PRINT command.	Y
PROmpt	Sets or displays prompt increment for linenumber editing.	Y
Quit	Terminates editing for a file.	Y

Figure 4-25 (Part 3 of 5). Summary of Editor Commands and Macros

COMMAND	FUNCTION	DBCS SUPPORT
RENum	Puts new sequence numbers into the file being edited.	Y
REPEAT	Executes a subsequent OVERLAY, BLANK, ALIGN, CENTER, JUSTIFY or SHIFT command a given number of times.	N
REPlace	Replaces the contents of a library member.	Y
REStore	Restores various editor settings from previous STACK EDIT.	Y
Rewrite	Replaces the current line with a specified string (see]REWRITE Command≠ on page B–8).	N
RIght	Shifts (the 'view' is shifted left) the data within the logical screen to the right.	Y
RPT	See REPEAT command.	N
SAve	Saves the contents of the input area as a member of the library.	Y
SCFeen	Defines and displays the number and size of logical screens.	Y
Search	Searches the file from the top for a given string of characters.	Y
SET	Sets various functions on or off. See also [/]SET system command.	Y
SHIft	Shifts data within the zone left or right a given number of columns.	N
SHow	Displays names of files being edited. See also [/]SHOW system command.	Y
SPlit	Splits the current line into two lines.	N
STACk	Places data or commands in the editor stack.	Y
STATUS	See [/]SHOW system command.	Y
TABset	See [/]TABSET system command.	Y

Figure 4-25 (Part 4 of 5). Summary of Editor Commands and Macros

COMMAND	FUNCTION	DBCS SUPPORT
Top	Positions the line pointer to the null line in front of the first line in the file.	Y
TYpe	See PRINT command.	Y
Up	Moves the line pointer upwards a given number of lines.	Y
Verify	Alters screen parameters.	Y
VIEW	Indicates how data is to be displayed on the screen.	Y
Zone	Sets the current zone for editing, or scanning operations.	Y
'nn'	Replaces or locates lines by sequence number.	Y

Figure 4-25 (Part 5 of 5). Summary of Editor Commands and Macros

ADD Command

The ADD command is used to add a string of characters behind the last non-blank character in the edit zone of the current line.

```
Add           string|/string/
```

string is the string of characters to be added to the line. If the string contains any commas, parentheses or blanks, delimiter characters must be used.

The string is added behind the last non-blank character of the current line (see example). If the string does not fit into the edit zone, it gets truncated. If there is no space left message `*MSG=FUNCTION REQUESTED BEYOND ZONE` is displayed.

The line pointer remains unchanged, the cursor is set to the command area.

Note: You can display the delimiter character with the SHOW command. You can change it with the DELIM command.

Example:

Add the string ',XREF' in the /OPTION statement.

```
===> add ,xref_
<<..+....1....+....2....+....3.  .5....+.. MEM=ASSEMJOB>>..+..FS
// OPTION DECK                                            /===/*
// EXEC ASSEMBLY                                          *===*
```

```
===>
<<..+....1....+....2....+....3.  .5....+.. MEM=ASSEMJOB>>..+..FS
// OPTION DECK,XREF                                       /===/*
// EXEC ASSEMBLY                                          *===*
```

ALIGN Command

The ALIGN command is used in text editing to align both the left and right hand margins of lines. The right and left margins are determined by the current zone setting, or by a column suffix.

```
ALIgn   [INdent]
```

INdent bypasses left justification, for example where you want an indented margin at the beginning of a paragraph.

The column suffix Cnn can be used with this command.

Alignment is performed by left justifying the line within the zone and then inserting additional blanks between words where existing blanks occur until the line is also justified at the right hand margin.

The line pointer remains unchanged, the cursor is set to the command area.

If more than one line is to be aligned, the REPEAT command can be issued prior to the ALIGN command to specify the number of lines to be aligned.

Example:

Align two lines of text within the zone from column 1 to 22. Assume that ';' is the line end character.

```
===> zone 1 22;repeat 2;align;top_
<<..+....1....+....2....+....3....+..  ..5....+.. MEM=member   >>..+..FS
HOW ARE YOU TODAY?                                               /===/*
I AM FINE THANK YOU.                                             *===*
***** END OF FILE *****                                          *****
```

```
===> _
<<..+....1....+....2>>..+....3....+..  ..5....+.. MEM=member   >>..+..FS
***** TOP OF FILE *****                                          /***/
HOW   ARE  YOU  TODAY?                                           *===*
I AM  FINE  THANK YOU.                                           *===*
***** END OF FILE *****                                          *****
```

The ZONE command establishes the requested zone setting (see scale line). The REPEAT command defines the number of lines to be aligned.

ALTER Command

The ALTER command lets you change one character to another, and reference a character by its hexadecimal value. Any of the 256 EBCDIC combinations may be referenced.

```
                             ┌ ┐
                             │n│
ALter        char1 char2     │*│  [G]  │S│
                             │1│       │O│
                             └ ┘
```

The column suffix Cnn can be used with this command.

char1	is the character to be altered. It can be either a single character or a pair of hexadecimal digits (00 through FF).
char2	is the character to which char1 is to be altered. It can be either a single character or a pair of hexadecimal digits.
n|*	indicates the number of lines to be searched for char1. '*' indicates that all lines in the file beginning with the current line are to be searched. If this operand is omitted, only the current line is searched. The maximum number of lines that can be searched is 99999.
G	indicates global change. Every occurrence of char1 in the edit zones of the lines searched is altered. If 'G' is omitted, only the first occurrence of char1 in each searched line is altered.
S|O	indicates that changes are to be recorded in the stack successively ('S') or starting at the beginning of the stack ('O').

Char1 is searched within the edit zone of the specified lines starting with the current line. If the line pointer is positioned at end-of-file, the search is started in the first line. If char1 is found, it is replaced by char2.

The line pointer is positioned at the last line that was searched. The cursor is set to the command area.

Global changes, that is changes of a character throughout a member may incorporate many single changes. They can be best controlled by recording each change in the stack. The stack must have been previously allocated either via the STACK OPEN command, or earlier through use of the punch area. The command PRINT $$STACK can be used to print the stack for viewing of the changes recorded.

Example:

You want to add a REP statement to an object deck. REP statements must start with X'02' in column 1. Because you cannot enter this character directly from a terminal keyboard, first enter ZREP, then alter Z to X'02'.

Note: Another possibility to do such a change would be to view the data in hexadecimal format (see "VIEW Command" on page 4-132) and then enter '02' directly.

```
===> l /rld /;insert zrep  0000e8 00147f0;alter z 02_
<<..+....1....+....2....+....3. .5....+.. MEM=TPROGOBJ>>..+..FS
***** TOP OF FILE *****                                 /***/
CATALOG TPROG.OBJ         EOD=/+          REPLACE=YES    *===*
 ESD            TPROG           IJ2L0010                 *===*
 TXT              K  Ys K    h&        Os          s  1  *===*
 TXT              30        K     .     0 ¢        0     *===*
 TXT                                                     *===*
 TXT                                     &               *===*
 TXT                                                     *===*
 TXT    y       HERE IS THE TEST PROGRAM                 *===*
 RLD                                                D    *===*
 END                                                     *===*
/+                                                       *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=TPROGOBJ>>..+..FS
 REP  0000E8 00147F0                                    /===/*
 END                                                    *===*
/+                                                      *===*
/*                                                      *===*
***** END OF FILE *****                                 *****
```

BACKWARD Command

The BACKWARD command allows 'backward' scrolling through a file.

```
BAckward     [nn|1]
```

nn is a decimal number greater than zero that represents the number of logical pages to be paged backward.

The BACKWARD command moves the line pointer backward (toward the top of the file) 'nn' pages. A page is defined as the number of lines contained in the format area of the screen. The cursor is set to the command area.

Notes:

1. *When the line pointer is less than 'nn' pages away from the top of the file, it is set to the first line in the file and the 'INVALID-RANGE' message is issued.*

2. *To scroll backward through a file one page at a time, precede the 'BA 1' command with an ampersand (&). The ampersand will cause the command to be retained in the command area. Thus, each time you press ENTER your display moves back one page.*

Example:

Page backward 5 logical screens:

```
    BA 5
```

Page backward 1 page and retain the command:

```
    &BA 1
```

BLANK Command

This command places blanks in the current line wherever nonblank characters occur in the mask. The logical tab character (unless IMAGE OFF is set), can be used to generate blanks in the mask operand.

```
BLank        mask
```

The column suffix Cnn can be used with this command.

mask is any valid input string.

The mask is separated from the command by only one blank. All other blanks are considered a part of 'mask'.

The REPEAT command can be entered before the BLANK command to extend the effect to more than one line.

Example:

Place blanks in columns 73-80 thus eliminating the sequence numbers.

```
===> repeat *;blankc73 ________;top_
<<..+....1....+....2....+....3. .5....+.. MEM=COMMANDS ...+..F>
***** TOP OF FILE *****                                    /***/
ASSUME THIS MEMBER CONTAINS SOME TEXT DESCRIBING           000100
THE VARIOUS COMMANDS OF THE VSE/ICCF EDITOR.  BY           000200
MISTAKE, IT HAS BEEN SERIALIZED, AND WE SHALL USE          000300
THE "BLANK" COMMAND TO ERASE THE SERIAL NUMBERS            000400
TEXT TEXT ................................. TEXT           000500
TEXT TEXT ................................. TEXT           000800
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=COMMANDS ...+..F>
***** TOP OF FILE *****                                    /***/
ASSUME THIS MEMBER CONTAINS SOME TEXT DESCRIBING
THE VARIOUS COMMANDS OF THE VSE/ICCF EDITOR.  BY
MISTAKE, IT HAS BEEN SERIALIZED, AND WE SHALL USE
THE "BLANK" COMMAND TO ERASE THE SERIAL NUMBERS
TEXT TEXT ................................. TEXT
TEXT TEXT ................................. TEXT
```

BOTTOM Command

The BOTTOM command positions the line pointer past the last line of the file.

```
Bottom
```

Use the BOTTOM command followed by the INPUT or INSERT command to add new lines at the end of the file.

Example:

Position the line pointer to the bottom of the file. Inserts (or input mode data) can now be entered and will be added beyond the last line in the file.

```
===> b_
<<..+....1....+....2....+....3. .5....+.. MEM=EXAMPLE >>..+..FS
***** TOP OF FILE *****                                   /***/
FIRST LINE                                                *===*
LAST LINE                                                 *===*
***** END OF FILE *****                                   *****
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=EXAMPLE >>..+..FS
LAST LINE                                                 *===*
***** END OF FILE *****                                   *****
```

CANCEL Command

The CANCEL command terminates the editor immediately and returns the terminal to command mode.

```
  CANcel
```

The CANCEL command has the same effect as the QUIT command on each active file. The CANCEL command is forced if the 3270 PA2[1] (cancel) key is pressed.

Example:

```
===> cancel_
<<..+....1....+....2....+....3.  .5....+.. MEM=EXAMPLE >>..+..FS
***** TOP OF FILE *****                                     /***/
FIRST LINE                                                  *===*
LAST LINE                                                   *===*
*****
```

```
_
...+....1....+....2....+....3.  .5....+....6....+....7....+..FS
*FULL SCREEN EDITOR TERMINATED
*READY
```

1 PA2 is the Cancel key in the distributed VSE/ICCF. Your installation may however have chosen PA1 or PA3 as the Cancel key.

CASE Command

The CASE command is used to control how upper and lower case translation is handled during editing.

```
CAse          [M|U]
```

M causes the editor to accept input of both upper and lower case data.

U causes the editor to return to normal upper case translation. If, however, the DBCS attribute is set for a member, this command will be rejected and the member will be treated as if case were set to mixed.

VSE/ICCF normally translates all input data to upper case. The CASE command alters this translation so that lower case data can be entered.

The CASE command only applies to the current edit session. At the end of the session, case translation reverts to what you were using when you started the edit session. (See the /SET CASE command.) In other words, if mixed case translation was in effect when the editor was entered, it will continue to be in effect until CASE U is specified. If the edit session is now terminated, the translation will revert back to mixed case translation.

If the CASE command is entered with no operand, the current CASE setting is displayed. (CASE=S on the display indicates that no CASE setting has been specified, so the system CASE setting is in effect. That is, the translation that you were using when the editor was entered is still in effect.)

If your display does not show lower case (3270), you can still enter lower case data if CASE=M is in effect. Note, however, that this data will not be displayed as lower case. Be aware of the problems that this might cause. For example, you might try to locate a string of characters that only **appear** to be in upper case.

CENTER Command

The CENTER command centers data within the edit zone of the current line.

```
CENter        [INdent]
```

The column suffix Cnn can be used with this command.

INdent can be used to cause any leading blanks within the zone to be considered as part of the data for centering.

If more than one line is to be centered, enter the REPEAT command prior to the CENTER command to specify the number of lines to be centered.

Example:

```
===> zone 1 30;repeat 2;center;top_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                   /***/
VSE/ICCF                                                  *===*
TERMINAL USER'S GUIDE                                     *===*
                                                          *****
```

```
===> _
<<..+....1....+....2....+...>>. .5....+.. INP=*INPARA* ...+..FS
***** TOP OF FILE *****                                   /***/
          VSE/ICCF                                        *===*
    TERMINAL USER'S GUIDE                                 *===*
```

CHANGE Command

The CHANGE command replaces one string of characters with another.

```
Change      [/string1/string2/ [n|*|1] [G] [S|O]]
```

The column suffix Cnn can be used with this command.

no operand the cursor is set to the current line.

/	is any unique delimiting character (not alphabetic or numeric) that does not appear in string1 or string2. The ending delimiter on string2 need not be specified if no operands follow (unless string2 contains trailing blank characters).
string1	is the character string to be replaced (old data).
string2	is the character string to replace string1 (the new data).
n|*	specifies the number of lines to be searched for string1. '*' indicates that all lines in the file beginning with the current line are to be searched. If this operand is omitted, only the current line is searched. The maximum number of lines that can be searched is 99999.
G	indicates global change. Every occurrence of string1 in the edit zones of the lines searched is changed. If 'G' is omitted, only the first occurrence in each searched line is changed.
S|O	indicates that changes are to be recorded in the stack successively ('S') or starting at the beginning of the stack ('O').

String1 is searched within the edit zone of the specified lines starting with the current line. If the line pointer is positioned at the end-of-file, the search is started in the first line. If string1 is found, it is replaced by string2. The result is truncated so that it fits into the edit zone. The strings can be of different length. String2 may be a null string. If string1 is not found nothing is altered.

The CHANGE command also processes mixed data. Characters in string1 and string2 may be combined in the following way:

String 1	**String 2**
One-byte characters	One-byte characters Mixed data Double-byte characters
Mixed data	Mixed data Double-byte characters
Double-byte characters	Double-byte characters

The CHANGE operation is rejected if:

- A subfield of double-byte characters specified in string2 would exceed the zone.

- The difference between the number of bytes of string1 and the number of bytes of string2 is not even and the double-byte characters specified in string1 are part of a subfield that extends beyond the zone column. (When you determine the number of bytes that make up string1 or string2, do not count a SO or SI control character, if it occupies the leftmost or rightmost position in the string.)

If the change yields a subfield of double-byte characters that has a length of zero, the adjacent SO and SI control characters are not removed.

The line pointer is positioned at the last line that was searched. The cursor is set to the command area (with the exception of a CHANGE command without operands).

Global changes, that is changes of a string throughout a member, may incorporate many single changes. They can be best controlled by recording each change in the stack. It must have been previously allocated either via the STACK OPEN command or by the prior use of the punch area. The command PRINT $$STACK can be used to print the stack for viewing of changes recorded.

Notes:

1. *The CHANGE command can be used to display, without changing, the first 100 lines (or more than 100 if the PUNCH/STACK area was greater than 100 lines when the editor was entered) that contain the information specified in string1. Enter:*

   ```
   change /string1/string1/ * 0
   print $$stack
   ```

2. *Use the ALTER command or OVERLAYX command to change a single character to some special character (one that is not available on your keyboard). The CHANGE command can be used to alter them to displayable characters.*

3. *No tab character should be in string1 or string2.*

Examples:

Example 1:

Change (globally) all occurrences of 'SAV%' into 'SAV$'.

```
===> c /sav%/sav$/ *;locup /consdir/_
<<..+....1....+....2....+....3. .5....+.. MEM=TPROG    >>..+..FS
            USING CONSDIR,R10                               /===/*
            LR    R10,R15                                   *===*
            ST    R13,SUBSAV%4+4                            *===*
            LA    R13,SUBSAV%4                              *===*
            PUT   CONSOLO                                   *===*
            L     R13,SUBSAV%4+4                            *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=TPROG    >>..+..FS
            USING CONSDIR,R10                               /===/*
            LR    R10,R15                                   *===*
            ST    R13,SUBSAV$4+4                            *===*
            LA    R13,SUBSAV$4                              *===*
            PUT   CONSOLO                                   *===*
            L     R13,SUBSAV$4+4                            *===*
```

Example 2:

This is another global change. A 'g' is specified to make the change effective for all occurrences in a line.

```
===> c /buff/buf1/ * g;locup /clrbuf/
<<..+....1....+....2....+....3. .5....+.. MEM=CODE1    >>..+..FS
CLRBUF   MVI   BUFF,C' '                                   /===/*
         MVC   BUFF+1(79),BUFF                             *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=CODE1    >>..+..FS
CLRBUF   MVI   BUF1,C' '                                   /===/*
         MVC   BUF1+1(79),BUF1                             *===*
```

Example 3:

Globally eliminate 'VSE/' from the file and open the stack to record all changes. '!' is used as delimiter character; '/' would not work because it is part of the change argument.

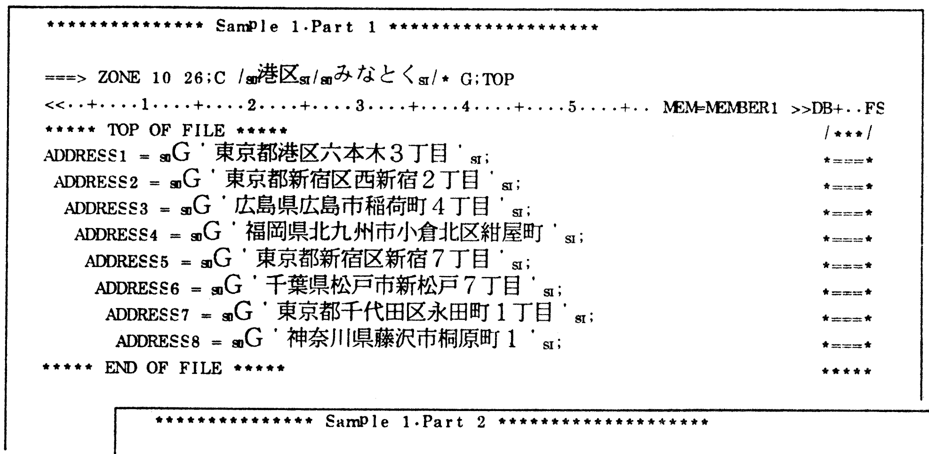
```
===> change !VSE/!! * g o;top_
<<..+....1....+....2....+....3. .5....+.. MEM=COMMANDS>>..+..FS
***** TOP OF FILE *****                                    /***/
This VSE/ICCF member contains some text describing         *===*
VSE/ICCF and the VSE/ICCF editor. In order to              *===*
make it more readable, the prefix "VSE" shall be           *===*
removed. We shall do this with a global change.            *===*
```

```
===> print $$stack_
<<..+....1....+....2....+....3. .5....+.. MEM=COMMANDS>>..+..FS
***** TOP OF FILE *****                                    /***/
This ICCF member contains some text describing             *===*
ICCF and the ICCF editor. In order to                      *===*
make it more readable, the prefix "VSE" shall be           *===*
removed. We shall do this with a global change.            *===*
```

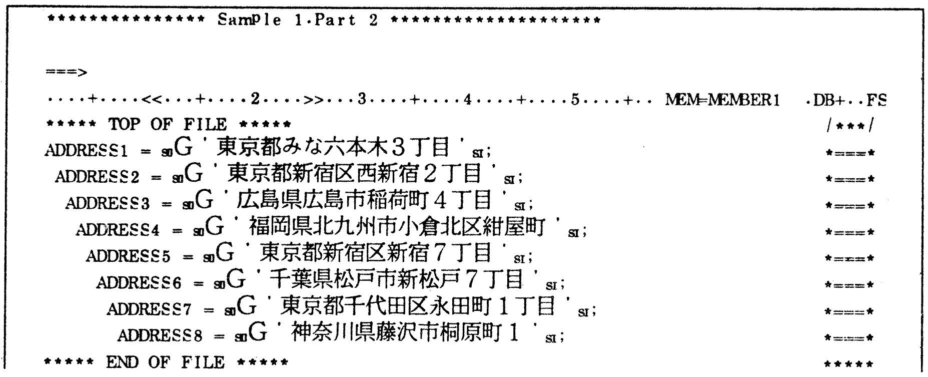
```
_
 ...+....1....+....2....+....3. .5....+....6....+....7....+..ED
This ICCF member contains some text describing
ICCF and the VSE/ICCF editor. In order to
ICCF and the ICCF editor. In order to
*END PRINT
```

Example 4:

Within the specified zone, replace all occurrences of the two double-byte characters specified in string1 by the four double-byte characters specified in string2. The characters specified in string1 are located on the first line and modified. Because string2 would exceed the zone column, it is truncated.



```
*************** Sample 1.Part 1 ********************

===> ZONE 10 26;C /so港区si/soみなとくsi/* G;TOP
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                                 /***/
ADDRESS1 = soG '東京都港区六本木３丁目' si;                              *===*
 ADDRESS2 = soG '東京都新宿区西新宿２丁目' si;                           *===*
  ADDRESS3 = soG '広島県広島市稲荷町４丁目' si;                          *===*
   ADDRESS4 = soG '福岡県北九州市小倉北区紺屋町' si;                     *===*
    ADDRESS5 = soG '東京都新宿区新宿７丁目' si;                          *===*
     ADDRESS6 = soG '千葉県松戸市新松戸７丁目' si;                       *===*
      ADDRESS7 = soG '東京都千代田区永田町１丁目' si;                    *===*
       ADDRESS8 = soG '神奈川県藤沢市桐原町１' si;                       *===*
***** END OF FILE *****                                                 *****
```



```
*************** Sample 1.Part 2 ********************


===>
....+....<<...+....2....>>...3....+....4....+....5....+.. MEM=MEMBER1  .DB+..FS
***** TOP OF FILE *****                                                 /***/
ADDRESS1 = soG '東京都みな六本木３丁目' si;                              *===*
 ADDRESS2 = soG '東京都新宿区西新宿２丁目' si;                           *===*
  ADDRESS3 = soG '広島県広島市稲荷町４丁目' si;                          *===*
   ADDRESS4 = soG '福岡県北九州市小倉北区紺屋町' si;                     *===*
    ADDRESS5 = soG '東京都新宿区新宿７丁目' si;                          *===*
     ADDRESS6 = soG '千葉県松戸市新松戸７丁目' si;                       *===*
      ADDRESS7 = soG '東京都千代田区永田町１丁目' si;                    *===*
       ADDRESS8 = soG '神奈川県藤沢市桐原町１' si;                       *===*
***** END OF FILE *****                                                 *****
```

Example 5:

Within the specified zone, replace all occurrences of the two double-byte characters specified in string1 by the one-byte and the double-byte character specified in string2. No data are modified, because the difference between the number of bytes of string1 (8 bytes) and the number of bytes of string2 (7 bytes) is not even and string1 is part of a subfield that extends beyond the zone column.

```
*************** Sample 2.Part 1 ********************

===> ZONE 10 30;C /= soG ' si/= Gso ' si/* G;TOP
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                             /***/
ADDRESS1 = soG ' 東京都港区六本木３丁目 ' si;                            *===*
 ADDRESS2 = soG ' 東京都新宿区西新宿２丁目 ' si;                          *===*
  ADDRESS3 = soG ' 広島県広島市稲荷町４丁目 ' si;                         *===*
   ADDRESS4 = soG ' 福岡県北九州市小倉北区紺屋町 ' si;                    *===*
    ADDRESS5 = soG ' 東京都新宿区新宿７丁目 ' si;                         *===*
     ADDRESS6 = soG ' 千葉県松戸市新松戸７丁目 ' si;                      *===*
      ADDRESS7 = soG ' 東京都千代田区永田町１丁目 ' si;                   *===*
       ADDRESS8 = soG ' 神奈川県藤沢市桐原町１ ' si;                      *===*
***** END OF FILE *****                                             *****
```

```
*************** Sample 2.Part 2 ********************

===>
....+....<<...+....2....+...>>....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                             /***/
ADDRESS1 = soG ' 東京都港区六本木３丁目 ' si;                            *===*
 ADDRESS2 = soG ' 東京都新宿区西新宿２丁目 ' si;                          *===*
  ADDRESS3 = soG ' 広島県広島市稲荷町４丁目 ' si;                         *===*
   ADDRESS4 = soG ' 福岡県北九州市小倉北区紺屋町 ' si;                    *===*
    ADDRESS5 = soG ' 東京都新宿区新宿７丁目 ' si;                         *===*
     ADDRESS6 = soG ' 千葉県松戸市新松戸７丁目 ' si;                      *===*
      ADDRESS7 = soG ' 東京都千代田区永田町１丁目 ' si;                   *===*
       ADDRESS8 = soG ' 神奈川県藤沢市桐原町１ ' si;                      *===*
***** END OF FILE *****                                             *****
```

```
*************** Sample 3.Part 1 ********************
```

@COPY Editor Macro

@MOVE Editor Macro

The @COPY and @MOVE editing macros copy or move up to 99 lines from one part of the file to another.

```
@COPY        nn          command   [operand]
@MOVE        /string/
```

nn	is a decimal number from 1 to 99 indicating the number of lines to be copied or moved.
/string/	is a character string defining the end of the lines to be copied or moved. All lines from the line pointer down to (but not including) 'string' (but not more than 99) will be copied or moved. The string must be bounded by the delimiter character and contain no blanks.
command	is an editing command which will be used to position the line pointer at the line after which the copied or moved data is to be inserted. Valid commands are: DOWN, FORWARD, LOCATE, LOCNOT, LOCUP, NEXT, POINT, TOP, UP or 'nn'.
operand	is the operand associated with the pointer movement command. This can be a decimal number (as in UP 3) or a string (as in LOC /XREF/). If this operand is a string, it must be bounded by the delimiter character and must contain no blank characters.

The @COPY and @MOVE editing macros are similar, except that the @MOVE macro deletes the specified lines from their former location whereas @COPY does not.

The 'command' and 'operand' operands together form an editing command that sets the line pointer to the place where the data is to be inserted. Before the target-setting command is executed, the position of the line pointer is

```
for @COPY   -  your actual current line;
for @MOVE   -  the next line which is not going to be moved (because
               the lines to be moved are first deleted).
```

The last line copied or moved becomes the new current line.

Notes:

1. *The @COPY and @MOVE macros use the stack area; therefore the previous contents of the stack area are lost.*
2. *The @COPY or @MOVE macro should not be executed from the stack.*
3. *If the destination (command operand) is incorrect, @MOVE can cause lines to be deleted from your file. You can recall them using the GETFILE $$STACK command.*

Examples:

The first example is shown with screen images. Three separator lines are to be copied behind the first line that contains, in columns 16-72, the string 'LOOPA'.

```
===> @copy 3 locatec16 /loopa/;up 3_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
******************************************************** /===/*
*                                                      * *===*
******************************************************** *===*
MAINPGM  CSECT                                           *===*
         PRINT GEN                                       *===*
         BALR  5,0                                       *===*
         USING *,5                                       *===*
         OPEN FILOUT                                     *===*
LOOPA    EXCP  RDCCB                                     *===*
         WAIT  RDCCB                                     *===*
         CLC   R(3),=C'/* '                              *===*
         BE    ENDCARD                                   *===*
         MVC   0(80,2),R                                 *===*
         PUT   FILOUT                                    *===*
         B     LOOPA                                     *===*
ENDCARD  CLOSE FILOUT                                    *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
         B     LOOPA                                     /===/*
******************************************************** *===*
*                                                      * *===*
******************************************************** *===*
ENDCARD  CLOSE FILOUT                                    *===*
```

The remaining examples focus on the command specification only; screen images are not shown.

1. Copy 3 lines beginning at the line pointer and place these lines 7 lines further down in the file.

   ```
   @copy 3 down 7
   ```

2. Copy 5 lines beginning at the line pointer and insert these lines after the first line containing string XREF.

   ```
   @copy 5 loc /XREF/
   ```

3. Make a copy of all the lines down to (but not including) the first occurrence of the string PHASE2 and put this copy after the first line encountered (proceeding upward in the file) containing the string INSRTX.

   ```
   @copy /PHASE2/ lup /INSRTX/
   ```

4. Copy 10 lines to the top of the file.

   ```
   @copy 10 top
   ```

5. Make a copy of all lines down to the string STOP and put this copy after the line with sequence number 127100 (assuming linenumber editing is in effect).

   ```
   @copy /STOP/ 127100
   ```

6. Move three lines beginning at the line pointer downward in the file seven lines.

   ```
   @move 3 down 7
   ```

7. Move 5 lines beginning at the line pointer after the first line encountered with the string XREF.

   ```
   @move 5 loc /XREF/
   ```

8. Move all the lines down to (but not including) the first occurrence of the string PHASE2 after the first line encountered (proceeding upward in the file) containing the string INSRTX.

   ```
   @move /PHASE2/ lup /INSRTX/
   ```

9. Move ten lines to the top of the file.

   ```
   @move 10 top
   ```

10. Move all lines down to the string STOP after the line with sequence number 127100 (assuming linenumber editing is in effect).

    ```
    @move /STOP/ 127100
    ```

CTL Command (see [/]SET System Command)

CURSOR Command

The CURSOR command is used to set the cursor to a particular position on the screen.

```
CURsor       CURrent
             INPut
             LINe [nn|1]
             TABBack [tt]
             TABForward [tt]
```

CURrent	advances the cursor to position 1 of the current line. If multiple 'current lines' are on the screen because of multiple logical screens (see the SCREEN command), or if multiple formats (see the FORMAT command) are in effect, the cursor will be advanced to the line associated with the screen or format with the most recent command activity (as indicated by the cursor position at the time the ENTER or PF key is pressed).
INPut	advances the cursor to the next (or only) command area on the screen. If the CURSOR INPUT command is equated to a PF key, pressing this key will set the cursor to the next command line. If there is no command line on the screen following the present cursor position, the cursor will be set to the first (or only) command line.
LINe	advances the cursor 'nn' lines and sets it to position 1 of the line. If the line advanced to is not a line of data, the cursor will be set to the next line. If the 'nn' value would cause the end of the screen to be exceeded, a 'wrap around' to the top of the physical screen will occur. Thus, if there are 21 lines on the physical screen, the command CURSOR LINE 16 would cause the cursor to be moved backwards 5 lines (21 minus 16). If the 'nn' operand is omitted, '1' line is assumed.
TABBack TABForward	moves the cursor backwards or forwards 'tt' columns from its present position. If 'tt' is omitted and if tab positions are set, the cursor will be moved backwards or forwards to the last or next tab position (see the "[/]TABSET Command" on page 3-143 for the setting of tab positions).

Notes:

1. *Because most of the CURSOR command options are relative to the current cursor position, the command options are most useful when equated to PF keys.*

2. *The '@FSEDPF' macro is available for setting PF keys to commands for use in full screen edit mode. Or use a logon macro instead (as described in "Invoking the Editor" on page 4-9).*

3. *It is possible to set a PF key to multiple cursor commands. For example, if a PF key is equated to 'CURSOR CUR#CURSOR TABF 40' where '#' is the logical line end character, pressing the PF key will cause the cursor to be positioned at column 41 of the line pointer record. Note that if '#' is already the logical line end character, you should precede it with an escape character when initially setting the PF key. In this way the line end character will not be interpreted when the 'SET PF' command is entered; it will only be interpreted when the actual PF key is pressed.*

4. *If a PF key is set to 'CURSOR TABF 0', it can be used instead of the ENTER key when you want to retain the cursor at its present location.*

Examples:

Set PF keys to some useful CURSOR command options:

```
SET PF4ED CUR INP           (set to next command line)
SET PF5ED CUR LINE 16       (set backwards 5 lines)
SET PF6ED CUR LINE 5        (set forward 5 lines)
SET PF7ED CUR CURRENT       (set to current line)
SET PF8ED CUR TABB 20       (set backwards 20 columns)
SET PF9ED CUR TABF 20       (set forward 20 columns)
```

DELETE Command

The DELETE command is used to delete a line from the file.

```
DELete        [n|*|1|/string/]
```

The column suffix Cnn is effective with the 'string' version of this command.

n	specifies the number of lines to be deleted. The default is 1 and the maximum is 9999.
*	specifies that all lines in the file from and including the current line through the end of the file, or to a maximum of 9999 lines, are to be deleted. However, an entire library member cannot be deleted in this manner.
/string/	specifies the string which, when matched, terminates the delete operation. The string delimiter must be specified in this form of the command.

If /string/ is specified, all lines, starting with the current line and up to (but not including) the first line in which /string/ is matched, are deleted.

If 'n' is specified, the DELETE command removes 'n' lines from the file, starting with the current line. Upon completion of this request, the pointer is positioned after the last deleted line.

If 'n' or /string/ is not specified, only the current line is deleted.

The response *EOF indicates that the end of file was reached by the command. To position the pointer at the top of the file, a TOP command must be issued.

Notes:

1. *You may also delete lines using the edit line command D; see the description on page 4-143.*
2. *If the logical screen is divided into more than one format area, use of the '*' or '/string/' operands sets all other format areas to the top of the file.*

Examples:

1. Delete the currenc line plus the next three lines of the file edited.

```
===> delete 4
<<..+....1....+....2....+....3. .5....+.. MEM=COBEXMP >>..+..FS
020500 WORKING STORAGE SECTION.                                *===*
020520 01 HELLO-MSG.                                           /===/*
020530    02 FILLER PIC X(6) VALUE 'HELLO '.                   *===*
020540    02 NAME PIC X(12) VALUE SPACES.                      *===*
020550    02 FILLER PIC X(15) VALUE 'HAVE A NICE DAY'.         *===*
030000 PROCEDURE DIVISION.                                     *===*
030010
...
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=COBEXMP >>..+..FS
020500 WORKING STORAGE SECTION.                                *===*
030000 PROCEDURE DIVISION.                                     /===/*
030010 ...                                                     /***/
```

The current line, plus the next three lines, are deleted. The line pointer is then positioned at the line following the last deleted line.

2. Delete all lines starting with the current line up to the line which contains the string /PROCEDURE/.

```
===> del /PROCEDURE/
<<..+....1....+....2....+....3. .5....+.. MEM=COBEXMP >>..+..FS
020500 WORKING STORAGE SECTION.                                *===*
020520 01 HELLO-MSG.                                           /===/*
020530    02 FILLER PIC X(6) VALUE 'HELLO '.                   *===*
020540    02 NAME PIC X(12) VALUE SPACES.                      *===*
020550    02 FILLER PIC X(15) VALUE 'HAVE A NICE DAY'.         *===*
030000 PROCEDURE DIVISION.                                     *===*
030010
...
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=COBEXMP >>..+..FS
020500 WORKING STORAGE SECTION.                                *===*
030000 PROCEDURE DIVISION.                                     /===/*
030010 ...                                                     /***/
```

All lines, starting with the current line and up to (but not including) the first record containing a /PROCEDURE/ sequence within the defined zone, are deleted.

DELIM Command

The DELIM command is used to set the string delimiter to a character other than a slash (/). It can also be used to return to the slash as delimiter character.

```
DELIM        char
```

char is any non-alphabetic, non-numeric character other than space, comma, right parenthesis, left parenthesis or one of the SET command control characters.

The SHOW command with no operand can be used to display the current delimiter setting.

This command is needed for the DELETE and STACK /string/ commands when the string itself contains a slash. In such a case the string operand cannot be delimited with a slash. The new delimiter remains in effect until the editor is terminated, or until another DELIM command is entered.

The delimiter character has a special use with the OVERLAY command. It can be used as a substitute blank character. That is, when used in an OVERLAY line, any delimiter character appearing in the line will cause a blank to overlay the corresponding position in the record being edited.

Example:

```
===> delim *_
<<..+....1....+....2....+....3. .5....+.. MEM=EXAMPLE >>..+..FS
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=EXAMPLE >>..+..FS
```

Now you could use for example the following command:

```
delete *// JOB*
```

DOWN Command (see NEXT Command)

DUP Command

The DUP command duplicates the current line 'n' times.

```
DUP          [n|1]
```

n specifies the number of times (from 1 to 1000) the current line is to be duplicated. If no value is specified, 1 is assumed.

After the duplication is completed, the line pointer indicates the last of the newly created lines.

Note: You may duplicate lines also with the editor line command "; see the description on page 4-144.

Example:

```
===> dup 2
<<..+....1....+....2....+....3. .5....+.. MEM=COBEXMP >>..+..FS
020500 WORKING STORAGE SECTION.                           *===*
020520 01 HELLO—MSG.                                      /===/*
020530    02 FILLER PIC X(6) VALUE 'HELLO '.              *===*
020540    02 NAME PIC X(12) VALUE SPACES.                 *===*
020550    02 FILLER PIC X(15) VALUE 'HAVE A NICE DAY'.    *===*
030000 PROCEDURE DIVISION.                                *===*
030010
...
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=COBEXMP >>..+..FS
020500 WORKING STORAGE SECTION.                           *===*
020520 01 HELLO—MSG.                                      *===*
020520 01 HELLO—MSG.                                      *===*
020520 01 HELLO—MSG.                                      /===/*
020530    02 FILLER PIC X(6) VALUE 'HELLO '.              *===*
020540    02 NAME PIC X(12) VALUE SPACES.                 *===*
030000 ...                                                /***/
```

ECHO Command (see [/] ECHO System Command)

END Command (see QUIT Command)

ENTER Command

The ENTER command allows you to access (edit), or create, additional files to the one (or more than one) that you are presently editing.

```
ENTer        [name [password]]
```

name	is the one to eight character name of the file to be edited. This can be an existing library member, the name of a new file to be created, $$PRINT or $$PUNCH.
password	need only be specified when entering a password protected library member for the first time.

The ENTER command is only used when you are editing more than one file at a time. "Recursive Editing" on page 4-25 explains the usage of the ENTER command.

If 'name' is not the file that you are presently editing (that is, you have issued no prior ENTER command for this name), 'name' is looked up in the library directory. The 'password' operand is only required for password protected members the first time you enter them with the ENTER command. If 'name' is not found within the main library, the editor assumes that you are creating a new file, and editing proceeds with the empty file.

If you have already edited the library member 'name' in the present session, editing resumes where it left off. That is, the line pointer, editing options, tabs, etc will be as they were.

If you do not specify 'name', it is assumed that you want to resume editing the input area. However, this is only possible if you first edited the input area via ED. You can nevertheless save the input area as a library member without leaving edit mode with the SAVE or FILE command.

When the ENTER command is issued, a logical screen is assigned to the entered file (see the SCREEN command). If a free logical screen is available, the file being entered will be displayed on that logical screen. If all logical screens are in use, the file being entered will use the logical screen associated with the command line on which the ENTER command is typed. The previous file (owner of the logical screen) will no longer appear on the physical screen. However, it can be brought back simply by issuing an ENTER command for that file. Note that simply because a file being edited does not appear on the physical screen (that is, does not currently own a logical screen) does not mean that it is no longer being edited. It is still being edited, and when a logical screen becomes available (via the ENTER command), you can carry on editing it at the point where you left off.

If you enter $$PRINT or $$PUNCH you should not have a job running asynchronously in an interactive partition. Any changes that you made in these areas would be overlaid by your interactive partition job. Remember, too, that the punch and stack areas are physically the same area. If you have used the stack area, explicitly or implicitly (as with the @MOVE macro or the line command 'M'), the previous contents of the punch area would be overlaid.

Notes:

1. *Any number of files can be edited concurrently; however, a maximum of eight can appear on the physical screen at any given time.*

2. *The SHOW NAMES command can be used to display the names of the files that you are editing, that is, the names previously specified on ENTER commands.*

3. *When the print area is entered, VIEW and ZONE are set to begin at column 3.*

4. *Since the punch and stack areas are physically the same, changes or replacements can be made to data already in the stack by entering the punch area via the ENTER command and making the changes.*

Example:

While editing member NEW, you want to temporarily look at member OLD.

```
===> enter old
<<..+....1....+....2....+....3. .5....+.. MEM=NEW     >>..+..FS
        DC     C'IJSYS01'                                /===/*
        ORG                                              *===*
        END                                              *===*
```

```
===> enter new_
<<..+....1....+....2....+....3. .5....+.. MEM=OLD     >>..+..FS
***** TOP OF FILE *****                                  /***/
OLDPGM   CSECT                                           *===*
         PRINT GEN          SHOW MACRO EXPANSIONS        *===*
         BALR  5,0          ESTABLISH ADDRESSABILITY     *===*
         USING *,5                                       *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=NEW     >>..+..FS
        DC     C'IJSYS01'                                /===/*
        ORG                                              *===*
        END                                              *===*
```

FILE Command

The FILE command saves the file that you are editing in the library and also releases it. FILE is thus the same as a SAVE command followed by QUIT.

```
FILe        ┌name [password] ┌PRIV┐┐  [DBCS=ON|OFF]
            │                │PUBL││
            └                └    ┘┘
```

name is the 1 to 8 character name, beginning with an alphabetic character, of the file to be saved in the library.

password is a 4 character password which can be specified when initially saving a file in the library.

PRIV/PUBL allows you to specify that the new member is to be saved in the library as private or public. If neither operand is specified, the private or public status will be set according to the default in your user profile.

DBCS=ON|OFF allows you to save the new member with the DBCS attribute. If the DBCS operand is not specified, the member is saved with the attribute that was set during the editor session by a SET DBCS command. If a SET DBCS command has not been specified either, DBCS=OFF is assumed for the new member.

If no other file is being edited, the editor terminates normally and command mode is entered. If other files are being edited (via ENTER commands), the logical screen associated with the file being filed will be released for use by another file.

If the FILE command is used for the input area, the 'name' operand must be specified and the input area will be saved in the library under this name.

If the FILE command is used for a newly created file (see ENTER command), the 'name' operand is not required. The file will be saved in the library using the name specified on the ENTER command. If a 'name' operand is specified, the operand will override the name specified in the ENTER command.

If the FILE command is used for a file which is already in the library, the 'name' operand must not be specified. A 'save' operation will not be performed since none is required. Thus, when the FILE command is issued for an existing library member, it is equivalent to a QUIT.

Example:

```
===> file
<<..+....1....+....2....+....3. .5....+.. MEM=COBEXMP >>..+..FS
020500 WORKING STORAGE SECTION.                              *===*
020520 01 HELLO-MSG.                                         /===/*
020530    02 FILLER PIC X(6) VALUE 'HELLO '.                 *===*
020540    02 NAME PIC X(12) VALUE SPACES.                    *===*
020550    02 FILLER PIC X(15) VALUE 'HAVE A NICE DAY'.       *===*
030000 PROCEDURE DIVISION.                                   *===*
030010
...
```

```
_
...+....1....+....2....+....3. .5....+....6....+....7....+..CM
*FULL SCREEN EDITOR TERMINATED.
*END PRINT
```

FIND Command

The FIND command causes a column-dependent comparison of the nonblank characters in 'string' with each line in the file.

```
Find        string
```

The column suffix Cnn can be used with this command.

string is any valid input line. It can contain blanks, double-byte characters, and the logical tab character.

The compare begins on the current line and continues down the file until a match occurs or until the end-of-file is reached. If an end-of-file condition immediately preceded the FIND command, an automatic TOP command is performed before FIND begins. If 'string' is found, the pointer is positioned to the record in which 'string' is contained. If 'string' is not found, the line pointer remains after the last line of the file. The compare is column-dependent: 'string' is matched against the leftmost columns of the zone only. That is, the search does not look for the occurrence of string in **any** columns of the line. Also, only columns that correspond to nonblank characters in 'string' take part in the comparison. For example, if 'string' is A C, the search will be for an A in the first column of the zone and for a C in the third column of the zone.

'String' is separated from the command by only one blank. All other blanks are considered part of 'string'. If 'string' includes double-byte characters and if the SO and/or SI control character(s) occupy the leftmost and/or rightmost position in it, they are not considered part of 'string'.

FIND can be used to search for a specific line identifier in columns 73-80 as long as the zone has been set to cover this area. One technique is to issue FIND with a C73 suffix.

When 'string' has been found, the located record is displayed. When the record is not found, the *EOF message is displayed. To position the pointer at the top of the file, a TOP command must be issued.

Non-column dependent scans are handled by

- the LOCATE command if you want to scan from a position within the file through end-of-file (see "LOCATE Command" on page 4-96);
- the SEARCH command if you want to scan through an entire file (see "SEARCH Command" on page 4-120).

Examples:

Example 1

Find a string starting in column 1.

```
===> f 030000_
<<..+....1....+....2....+....3. .5....+.. MEM=member  >>..+..FS
***** TOP OF FILE *****                                 /***/
 CBL APOST                                              *===*
000000 IDENTIFICATION DIVISION.                         *===*
000010 PROGRAM-ID. 'HELLO'.                             *===*
010000 ENVIRONMENT DIVISION.                            *===*
020000
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=member  >>..+..FS
030000 PROCEDURE DIVISION.                               /===/*
030010 START-OF-PROGRAM.                                 *===*
030020     DISPLAY 'ENTER YOUR FIRST NAME' UPON CONSOLE. *===*
030030     ...                                           *===*
```

Example 2

Use a logical tab character with the FIND command (for general information on the use of the logical tab character, see "Using Tab Stops for Column-Oriented Editing" on page 4-21).

```
===> f $PROCEDURE
<<..+....1....+....2....+....3. .5....+.. MEM=member  >>..+..FS
***** TOP OF FILE *****                                 /***/
 CBL APOST                                              *===*
000000 IDENTIFICATION DIVISION.                         *===*
000010 PROGRAM-ID. 'HELLO'.                             *===*
010000 ENVIRONMENT DIVISION.                            *===*
020000
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=member  >>..+..FS
030000 PROCEDURE DIVISION.                               /===/*
030010 START-OF-PROGRAM.                                 *===*
030020     DISPLAY 'ENTER YOUR FIRST NAME' UPON CONSOLE. *===*
030030     ...                                           *===*
```

Assuming that the logical tab settings are set to columns 8, 12, 16,... and the logical tab character is '$' then the request searches for PROCEDURE starting in column 8.

Example 3

Use the column suffix with the FIND command.

```
===> fc08 PROCEDURE
<<..+....1....+....2....+....3. .5....+.. MEM=member  >>..+..FS
***** TOP OF FILE *****                                  /***/
 CBL APOST                                               *===*
000000 IDENTIFICATION DIVISION.                          *===*
000010 PROGRAM-ID. 'HELLO'.                              *===*
010000 ENVIRONMENT DIVISION.                             *===*
020000
       ===> _
       <<..+....1....+....2....+....3. .5....+.. MEM=member  >>..+..FS
       030000 PROCEDURE DIVISION.                               /===/*
       030010 START-OF-PROGRAM.                                 *===*
       030020     DISPLAY 'ENTER YOUR FIRST NAME' UPON CONSOLE. *===*
       030030     ...                                           *===*
```

The request searches for the string PROCEDURE starting in column 8.

Example 4

Find the specified double-byte character starting in column 15.

```
*************** Sample 3-Part 1 ********************

===> FC15 soGsi
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                          /***/
ADDRESS1 = soG '東京都港区六本木３丁目' si;                      *===*
 ADDRESS2 = soG '東京都新宿区西新宿２丁目' si;                   *===*
  ADDRESS3 = soG '広島県広島市稲荷町４丁目' si;                  *===*
   ADDRESS4 = soG '福岡県北九州市小倉北区紺屋町' si;             *===*
    ADDRESS5 = soG '東京都新宿区新宿７丁目' si;                  *===*
     ADDRESS6 = soG '千葉県松戸市新松戸７丁目' si;               *===*
      ADDRESS7 = soG '東京都千代田区永田町１丁目' si;            *===*
       ADDRESS8 = soG '神奈川県藤沢市桐原町１' si;               *===*
***** END OF FILE *****                                          *****

     *************** Sample 3-Part 2 ********************

     ===>
     <<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
       ADDRESS3 = soG '広島県広島市稲荷町４丁目' si;                  /===/*
        ADDRESS4 = soG '福岡県北九州市小倉北区紺屋町' si;             *===*
         ADDRESS5 = soG '東京都新宿区新宿７丁目' si;                  *===*
          ADDRESS6 = soG '千葉県松戸市新松戸７丁目' si;               *===*
           ADDRESS7 = soG '東京都千代田区永田町１丁目' si;            *===*
            ADDRESS8 = soG '神奈川県藤沢市桐原町１' si;               *===*
     ***** END OF FILE *****                                          *****
```

FLAG Command

The FLAG command is used to record changes to a file.

```
FLag            [ON|OFF]
```

ON causes all later changes to the file being edited to be flagged in columns 73-80 (unless some other flagging column has been defined by the VSE/ICCF administrator in the tailoring macro DTSOPTNS).

OFF turns this feature off. The OFF operand can only be used by the VSE/ICCF administrator.

Certain library members may have flagging automatically set on each time the editor is entered. This is done by the /PROTECT command, which sets a permanent indicator in the library directory (which, in turn, forces editor change flagging).

If flagging is on, each changed record will be coded with an 8-character change indication. The first character indicates the following type of alteration:

```
A - addition to file
C - part of the record was changed
N - record replaced via line number editing
R - record was replaced
```

The next three characters indicate the date of the alteration. The format is 'MDD': 'M' (month) is 1 through 9 for January through September, or 'O', 'N' or 'D' for October, November or December. 'DD' is the day of the month. The last four characters identify the user who made the change.

If you are using both linenumber editing and flagging, and if the flag would occupy the same columns as the sequence number, the sequence number takes precedence. That is, flagging would not take place.

The 8-character change indicator can only be modified by the VSE/ICCF administrator. Editor change flagging should not be used for RPG source members.

Notes:

1. *Flagging might overlay every column in the record, depending on your installation's default for editor change flagging. To protect certain members, such as procedures, macros, RPG programs, DTSUTIL and DTSANALS jobs from flagging, contact your VSE/ICCF administrator.*

2. *The FLAG command is always directed to one particular file. Therefore, if the ENTER command is used to edit several different files concurrently, flagging is active only for any one of these files for which the FLAG command had been given.*

Example:

Assume user USR1 had entered the command

```
flag on
```

Then a change operation would cause the record to be flagged as in the following figure.

```
===> change /ar/xr/
<<..+OFFSET=008 ...2....+....3. .5....+.. MEM=FLAGMEM >>..+..FS
***** TOP OF FILE *****                                     /***/
 MVC   FLDAR,FLDBR                                          *===*
```

```
===>
<<..+OFFSET=008 ...2....+....3. .5....+.. MEM=FLAGMEM >>..+..FS
 MVC   FLDXR,FLDBR                                  CN30USR1 /***/
```

FORMAT Command

The FORMAT command lets you divide a logical screen into several format areas in order to edit or view a file on several places at the same time. It also displays the current setting of format areas.

```
   FORMat        [[*]m-n [[*]m-n ... [[*]m[-n]]...]]
```

*m-n	defines one format area. Up to 8 format areas may be defined.
*	indicates that the corresponding format area will not contain line command areas.
m	is a digit (1, 2, or 3) indicating the number of lines to be used for displaying each logical record.
n	is a decimal number no lower than 2 indicating the number of logical lines to be displayed within the format area.

If the FORMAT command is entered with no operands, the current format settings will be displayed in the first command area for the logical screen. With the SCREEN command you can divide the physical 3270 screen into logical screens which allows you to view and edit more than one file concurrently. With the FORMAT command you can split a logical screen into subsections so that different parts within a file can be edited and displayed at the same time.

The FORMAT command allows you to divide the logical screen into from 1 to 8 format areas. Each operand represents one format area. Each format area has its own line pointer and its own command area. The number of physical lines within this command area (usually 1 or 2) is determined by the VERIFY command. The first format area within a logical screen also contains the scale/header line for that logical screen.

With '*' you may suppress the line command area which increases the amount of data which can be displayed per line from 72 to 79 columns. However, you can no longer use the editor line commands. For hints on how to make the 72 data columns suffice for displaying your data, see "Editing All 80 Columns" on page 4-34.

With 'm' you specify that either 1, 2, or 3 physical screen lines are to be used to display a record. More than one line may be necessary if the VIEW specification makes the displayed record exceed 79 characters. For example, to display all 80 characters in hexadecimal format would require three physical lines per record.

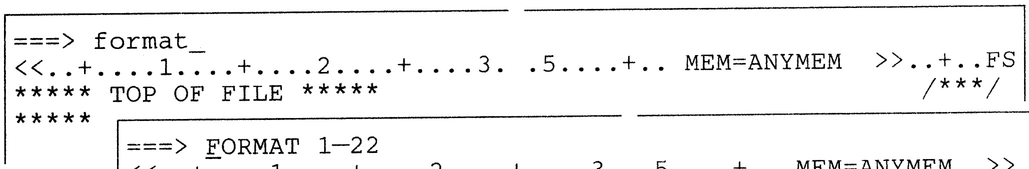
With '-n' you specify the size in number of logical lines of the format area. It can be omitted on the last operand, in which case the editor will calculate how many lines will fit into the remaining area of the logical screen and assign this value as the second operand of the pair.

Examples:

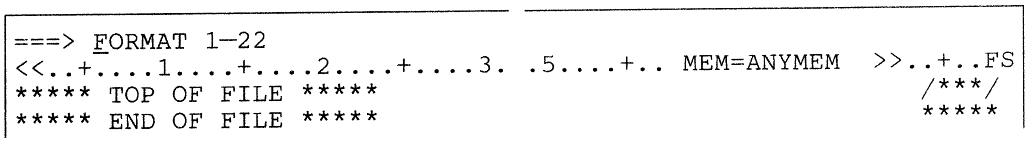
Example 1:

Display the currently active format settings.

Upon entry to the full screen editor, there is one logical screen (containing 24 lines for a 3270 Model 2) on the physical screen and one format area within the logical screen. The number of physical lines in the command area is set in the ED macro to 1 (this may have been changed for your installation). In addition, there is a scale/header line. Thus, the initial format specification is:



```
===> format_
<<..+....1....+....2....+....3. .5....+.. MEM=ANYMEM  >>..+..FS
***** TOP OF FILE *****                                  /***/
*****
```



```
===> FORMAT 1—22
<<..+....1....+....2....+....3. .5....+.. MEM=ANYMEM  >>..+..FS
***** TOP OF FILE *****                                  /***/
***** END OF FILE *****                                  *****
```

That is, a logical record is displayed on one line and there are 22 data lines (plus one scale line plus one line for the command area equals 24 lines).

Example 2:

Divide the logical screen into two format areas. The first format area allows ten records to be displayed while the second one can display as many as remain on the screen. Now you enter the FIND command in the first format area to display SUBPGM in the second format area.

```
===> format 1—10 1
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
LOOPB    GET   FILIN                                       /===/*
         LA    13,SAVEAREA                                 *===*
         LR    1,2                                         *===*
         CALL  SUBPGM                                      *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
LOOPB    GET   FILIN                                       /===/*
         LA    13,SAVEAREA                                 *===*
         LR    1,2                                         *===*
         CALL  SUBPGM                                      *===*
         LTR   1,1                                         *===*
         BZ    CARDOK                                      *===*
         MVC   P+85(12),=C'INVALID CARD'                   *===*
CARDOK   MVC   P(80),0(2)                                  *===*
         EXCP  PRCCB                                       *===*
         WAIT  PRCCB                                       *===*
===> find subpgm_
***** TOP OF FILE *****                                    /***/
/LOAD ASSEMBLY                                             *===*
/OPTION NOSAVE,LIST                                        *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
LOOPB    GET   FILIN                                       /===/*
         LA    13,SAVEAREA                                 *===*
         LR    1,2                                         *===*
         CALL  SUBPGM                                      *===*
         LTR   1,1                                         *===*
         BZ    CARDOK                                      *===*
         MVC   P+85(12),=C'INVALID CARD'                   *===*
CARDOK   MVC   P(80),0(2)                                  *===*
         EXCP  PRCCB                                       *===*
         WAIT  PRCCB                                       *===*
===> _
SUBPGM   CSECT                                             /===/*
         SAVE  (14,12)                                     *===*
         LR    5,15                                        *===*
         USING SUBPGM,5                                    *===*
         LA    4,5                                         *===*
         LR    3,1                                         *===*
LOOP     CLI   0(3),C'0'                                   *===*
```

For another application of the FORMAT command, see the second example under "VIEW Command" on page 4-132.

FORWARD Command

The FORWARD command allows you to scroll, or page, through the file that you are editing.

```
FOrward      [nn|1]
```

nn is a decimal number greater than zero representing the number of logical pages to be scrolled forward.

When the FORWARD command is entered, the line pointer is moved forward (toward the bottom of the file) by 'nn' pages. A logical 'page' consists of the lines displayed on the logical screen.

Notes:

1. *When the line pointer is less than 'nn' pages away from the bottom of the file, the line pointer is set to the last line in the file and the 'INVALID RANGE' message is issued.*
2. *To scroll forward through a file one page at a time, precede the 'FORWARD 1' command with the repeat prefix &.*

Examples:

Scroll forward 5 logical screens:

```
FORWARD 5
```

Scroll forward 1 page and retain the command:

```
&FOR 1
```

@FSEDPF Editor Macro

The @FSEDPF macro sets the editor set of PF keys for use.with the full screen editor. It is valid only in edit mode.

```
@FSEDPF
```

This macro sets certain PF keys to functions that are useful in full screen edit mode. The following commands are issued within the macro, unless your installation is using other settings:

```
SET PF1ED BACK 1             (Page backward 1 page)
SET PF2ED NEXT 10            (Page forward 10 lines)
SET PF3ED FORW 1             (Page forward 1 page)
SET PF4ED CURSOR INP         (Set cursor to next command line)
SET PF5ED CURSOR LINE 16     (Set cursor forward 16 lines)
SET PF6ED CURSOR LINE 5      (Set cursor forward 5 lines)
SET PF7ED CURSOR CUR         (Set cursor to current line)
SET PF8ED CURSOR TABB 20     (Set cursor backward 20 columns)
SET PF9ED CURSOR TABF 20     (Set cursor forward 20 columns)
SET PF10ED CURSOR TABF 0     (Leave cursor unchanged)
```

You can use the SET command to change the PF key settings.

GETFILE Command

The GETFILE command copies all or a portion of a library member or work area into the file that you are editing.

```
GETfile      name [password] [first|1 [num|*]]
             *
```

name	is the 1 to 8 character name of the library member to be copied. If the data to be copied is alphanumeric, 'name' can also be specified as one of the work areas: $$STACK (stack), $$PUNCH (punch area), $$PRINT (print area) or $$LOG (your log area). If mixed data is to be copied, 'name' can only refer to a member for which the DBCS attribute has been set and which is not a print-type member, or to the temporary work areas $$LOG, $$PUNCH, and $$STACK. In order to display the double-byte characters correctly, you first have to save the new member and set the DBCS attribute for it.
password	is a 1 to 4 character password, which need only be specified if the file being copied is password protected.
*	indicates that the data to be inserted is from another part of the member being edited.
first	is the number of the first record to be copied from the referenced member. The default is the whole file.
num	the number of lines to be copied. The default, or if specified as '*', is all lines from the line specified as first to end-of-file. The maximum number that you can specify as 'num' is 99999.

The GETFILE command can also be used to copy data within a file. The copied lines are inserted after the current line. 'name' can be specified as '*', which indicates that the data being copied is from the same file.

GETFILE can be used to copy data from the stack ($$STACK or $$PUNCH) into a file. For example, you can place data from one or more locations within a program into the stack using the STACK command. You can then re-insert this data into the program by specifying GETFILE $$STACK.

The last line copied always becomes the current line, unless the GETFILE command had been issued after a BOTTOM command. In this case, the line pointer would remain past end-of-file.

The *END INSERT message always indicates when a GETFILE operation is complete. The last line copied into the file is displayed.

If you copy (part of) a file into another area of the same file using GETFILE *, the copy operation stops one line before the current line. It thus avoids copying lines more than once. If you issue GETFILE * and the line pointer is positioned at top-of-file, the complete file is copied.

Example:

```
===> getfile member2 10 3;top_
<<..+....1....+....2....+....3. .5....+.. MEM=MEMBER1 >>..+..FS
RECORD  1  OF MEMBER1                                     /===/*
RECORD  2  OF MEMBER1                                     *===*
RECORD  3  OF MEMBER1                                     *===*
RECORD  4  OF MEMBER1                                     *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=MEMBER1 >>..+..FS
***** TOP OF FILE *****                                   /***/
RECORD  1   OF MEMBER1                                    *===*
RECORD NUMBER 10 OF MEMBER2                               *===*
RECORD NUMBER 11 OF MEMBER2                               *===*
RECORD NUMBER 12 OF MEMBER2                               *===*
RECORD  2   OF MEMBER1                                    *===*
RECORD  3   OF MEMBER1                                    *===*
RECORD  4   OF MEMBER1                                    *===*
```

HARDCPY Command (see [/]HARDCPY System Command)

INDEX Command

The INDEX command is used when editing large members to move the line pointer more easily from one area of the file to another.

```
INDex          [nn|100]
```

nn is a decimal number between 40 and 400 indicating the number of lines between entries in the index. If 'nn' is omitted, 100 will be assumed.

The INDEX command causes an index table to be built with a maximum of 31 entries. Each entry contains information that allows you to use the POINT command to quickly move to the indicated area of the file.

If you are using linenumber editing when you build the INDEX, the sequence numbers associated with index table pointers are also retained. This gives you a very handy way of rapidly moving the line pointer. To quickly move the line pointer to some distant area of the file, simply enter the sequence number of the line you want (see "'nn' Command" on page 4-140).

Indexed editing is very useful in members containing more than 500 lines, especially if your editing within the files is fairly random.

For members of about 3000 lines or larger, it is most efficient to specify the operand of the INDEX command as approximately 1/30th of the size of the file.

For linenumber editing it is important that the LINEMODE command be issued before the INDEX command. In this way the sequence numbers can be included in the index table.

There are a maximum of 31 index table entries available for a file. If the operand specified when building the index is less than 1/31st of the size of the file, it may take longer to position the line pointer to records toward the end of the file.

Example:

Assume you are editing a member that has sequence numbers in columns 73-78.

```
linemode 73 6
index
```

INPUT Command

The INPUT command allows you to insert new lines of data below the current line.

```
INPut
```

After the INPUT command has been entered, all lines in the format area below the current line are set to allow input of data, that is, they are set to unused blank or null lines. These lines are not actually a part of the file until they are filled-in and ENTER has been pressed.

Input lines are indicated on the screen by the characters '*INPUT' in the line command area. (Line commands can be entered in the line command area of input lines.) If the format area does not contain line command areas, the characters '*I*' will appear at the beginning of each input line. These characters will be overtyped as input data is entered.

After the INPUT command has been entered, the cursor will be set to the beginning of the first input line. The VSE/ICCF logical tabbing facility can be used for the entry of new data. Or, you can move the cursor to the correct point on the line and enter the appropriate data. Note that unless the NULLS option is set OFF (the default is ON), forward cursor movement within this area should be done with the space bar and not with the forward cursor (right arrow) key (see "SET Command" on page 4-122).

After all new data has been typed in, you should press ENTER. All new lines will be added to the file and the line pointer will be set to the last new line entered.

Notes:

1. *All input lines are treated as beginning in column 1 and ending in column 80. That is, the ZONE, LEFT, and VIEW specifications do not apply to input lines. However, once lines have been entered into the file and are redisplayed on the screen, they then become subject to these specifications.*

2. *The cursor movement commands (CURSOR, TABF, and TABB) should not be used while entering new data in the input area. Use logical tabbing instead.*

3. *If you want to enter more new lines than can be entered on one screen, enter an ampersand before the command (that is, &INPUT). Then each time the ENTER key is pressed, the lines entered will be added to the file and the screen will be once again formatted for more input lines.*

INSERT Command

The INSERT command inserts a single new line of data into the file.

```
  Insert      [string|/string/]
```

string is the string of characters to be inserted into the file. It can contain blanks, double-byte characters, and the logical tab character

The column suffix Cnn can be used with this command.

The string is inserted after the current line. The pointer is advanced to point to this inserted string (unless a BOTTOM command preceded the INSERT in which case the line pointer remains at the bottom). The string is separated from the command by only one blank; all other blanks are considered part of the string unless the string delimiter is used.

Without an operand, the INSERT command has the same effect as the INPUT command. The only exception is that, after the data has been entered, the line pointer remains at its original position (that is, it will not be set to the last line entered).

A blank line can be inserted into the file by using delimiter characters as follows

```
insert / /
```

You find another example of the INSERT command in Figure 4-22 on page 4-33.

JUSTIFY Command

The JUSTIFY command justifies data to either the left or right hand margin.

```
JUStify       [Left|Right]
```

Left   justification is to the left margin.

Right justification is to the right margin.

The column suffix Cnn can be used with this command.

The right and left margins are determined by the current zone setting or by a column suffix if present. Data within the zone can be shifted to either the left or right edge of the zone, thus squeezing out any intervening blanks.

If more than one line is to be margin justified, you can issue REPEAT prior to the JUSTIFY command to specify the number of lines to be justified.

If no operand is specified, 'left' is assumed.

Example:

```
===> zone 20 25;repeat *;justify right;top
<<..+....1....+....2....+....3. .5....+.. MEM=INVENTRY>>..+..FS
***** TOP OF FILE *****                                   /***/
ITEM1              5                                      *===*
ITEM2              28                                     *===*
ITEM3              2                                      *===*
ITEM4              2378                                   *===*
ITEM5              28355                                  *===*
ITEM6              289                                    *===*
```

```
===>
....+....1....+....<<..>>....3. .5....+.. MEM=INVENTRY ...+..FS
***** TOP OF FILE *****                                   /***/
ITEM1                   5                                 *===*
ITEM2                  28                                 *===*
ITEM3                   2                                 *===*
ITEM4                2378                                 *===*
ITEM5               28355                                 *===*
ITEM6                 289                                 *===*
```

LADD Command

The LADD command adds blank lines to the file.

```
LAdd          [nn|1]
```

nn is a decimal number (from 1 to 999) indicating the number of lines to be added to the file.

After the LADD command is entered, 'nn' lines are added to the file after the line and the cursor is set to the beginning of the first of these new lines. New data can then be typed into these blank lines.

Logical tabbing can be used within the new lines. If logical tab characters appear within the data on the screen, it is assumed that the line represents a complete 1 to 80 replacement of the blank line. If no tab characters appear within the newly entered data, the input will be interpreted according to your present VIEW ZONE, LEFT, and RIGHT specifications.

Unlike the INPUT command, which creates a screen of null lines for data entry, the LADD command creates only the specified number of blank lines to your file. If you add more blank lines than are needed, do not forget to delete the ones not required.

LEFT Command

The LEFT command shifts displayed data to the left within a logical screen.

```
LEft        [nn|1]
```

nn is a decimal number indicating the number of columns by which the data is to be shifted to the left.

It is often necessary to view columns of data which are not displayed on the screen. There are two ways to accomplish this. One is by using the VIEW command to specify that only certain columns are to be displayed (see the VIEW command). The second method is by shifting the displayed data left (or right) using the LEFT (or RIGHT) command.

When you issue the LEFT command, the data within the logical screen will be shifted 'nn' columns to the left. If 'nn' is omitted, one column is assumed. No data loss occurs and you have only to enter the RIGHT command to cause the data to be shifted back to its previous position.

The number of columns shifted to the left from the normal or base position is called the offset. If a non-zero offset exists, it will be displayed in the scale/header line as a reminder. A maximum offset of 149 is accepted. To return the display to the normal or base position, enter a RIGHT command for as many columns as are indicated in the offset.

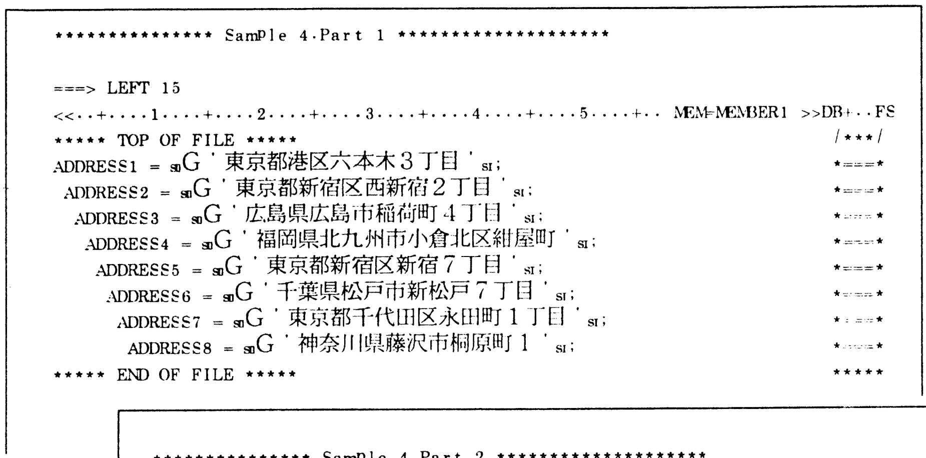
When you shift mixed data and when a double-byte character would occupy the leftmost position on the screen after the shift operation, this double-byte character will not be displayed. It is replaced by one or two control characters, which guarantee that the remainder of the subfield is displayed correctly. The data in the VSE/ICCF library is not affected by this operation, as long as you do not:

- Replace the added control characters
- Change double-byte characters to one-byte characters.

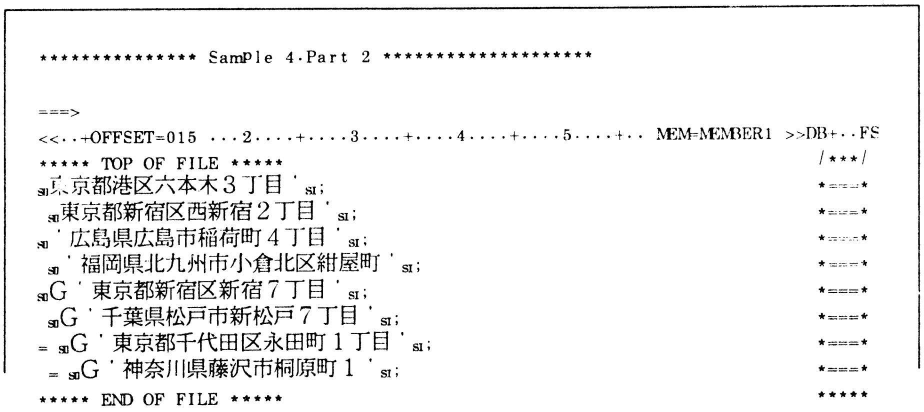
When you shift mixed data and an SI control character would occupy the leftmost position on the screen after the shift operation, the control character will be displayed as an unprintable character.

Example:

Shift mixed data 15 columns to the left.



```
*************** SamPle 4·Part 1 ********************

===> LEFT 15
<<··+····1····+····2····+····3····+····4····+····5····+·· MEM=MEMBER1 >>DB+··FS
***** TOP OF FILE *****                                                   /***/
ADDRESS1 = SOG '東京都港区六本木３丁目' SI;                                *===*
 ADDRESS2 = SOG '東京都新宿区西新宿２丁目' SI;                             *===*
  ADDRESS3 = SOG '広島県広島市稲荷町４丁目' SI;                            *===*
   ADDRESS4 = SOG '福岡県北九州市小倉北区紺屋町' SI;                       *===*
    ADDRESS5 = SOG '東京都新宿区新宿７丁目' SI;                            *===*
     ADDRESS6 = SOG '千葉県松戸市新松戸７丁目' SI;                         *===*
      ADDRESS7 = SOG '東京都千代田区永田町１丁目' SI;                      *===*
       ADDRESS8 = SOG '神奈川県藤沢市桐原町１' SI;                         *===*
***** END OF FILE *****                                                   *****
```



```
*************** SamPle 4·Part 2 ********************

===>
<<··+OFFSET=015 ···2····+····3····+····4····+····5····+·· MEM=MEMBER1 >>DB+··FS
***** TOP OF FILE *****                                                   /***/
SO東京都港区六本木３丁目' SI;                                              *===*
 SO東京都新宿区西新宿２丁目' SI;                                           *===*
SO '広島県広島市稲荷町４丁目' SI;                                          *===*
 SO '福岡県北九州市小倉北区紺屋町' SI;                                     *===*
SOG '東京都新宿区新宿７丁目' SI;                                           *===*
 SOG '千葉県松戸市新松戸７丁目' SI;                                        *===*
= SOG '東京都千代田区永田町１丁目' SI;                                     *===*
 = SOG '神奈川県藤沢市桐原町１' SI;                                        *===*
***** END OF FILE *****                                                   *****
```

LIBRARY Command (see [/]LIBRARY System Command)

LINEMODE Command

The LINEMODE command is used to start or terminate linenumber editing.

```
LINemode        ┌             ┐
                │n1 [n2|8]    │
                │Left         │
                │Right        │
                │Off          │
                └             ┘
```

n1	is a decimal number from 1 to 80 indicating the column number where the sequence number for linenumber editing begins.
n2	is a decimal number from 1 to 8 indicating the number of columns occupied by the sequence number. If 'n1' is specified but 'n2' is omitted, 'n2' will be assumed to be 8.
Left|Right	These operands can be specified instead of the exact sequence number location and length. If LEFT is specified, the sequence number is assumed to start at column 1 and to be 5 columns long. LINEMODE LEFT is useful for editing BASIC program files. If RIGHT is specified, the sequence number is assumed to start in column 73 and to be 8 columns long.
Off	Specifying OFF terminates linenumber editing.

To set linenumber editing on, specify as 'n1' and 'n2' the location and the length of the sequence number field within the records being edited. Once linenumber editing is in effect, the 'nn' command can be used to add, replace or locate lines based on this imbedded sequence number. When line number editing is set on, the default prompt increment for input mode will be set to 10. This value can be changed with the PROMPT command.

Notes:

1. *When you are prompted on a typewriter terminal, enter your input line on the same line as the prompted line number. In right-handed line number editing, the sequence numbers are not displayed in columns 73 to 80 (unless you use the VERIFY command to increase the verification setting). In line number editing on a display terminal, the prompting numbers in input mode appear on line 2 of the display output area. Enter your input lines in your input area.*

2. *If linenumber editing is set so that the sequence number field does not begin in column 1 or in column 73 or beyond, you cannot use the INSERT or REPLACE commands to insert a single line; use the 'nn' command instead.*

3. *When you initialize linenumber editing for files that already exist, the editor assumes that the records are in the proper format and numbered in ascending order.*

4. *The LINEMODE command may alter the zone setting if the sequence number field is at the beginning or the end of the input area and if the zone would have overlapped this area.*

Examples:

Example 1:

Type in a new member and set it up for linenumber editing. Example 2 shows how you actually use line numbers to your advantage.

```
===> tabset 1 6;set tab=¢;linemode left;input_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                    /***/
***** END OF FILE *****                                    *****
```

```
===> top_
....+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
***** TOP OF FILE *****                                    /***/
¢print 'enter three numbers separated with commas'        *INPUT
¢print 'enter /* to terminate'                            *INPUT
¢l = i + j + k                                            *INPUT
¢m = I * &sqr2                                            *INPUT
¢n = j * &pi                                              *INPUT
¢print i,j,k                                              *INPUT
¢print l,m,n                                              *INPUT
¢print 'end of calculation'                               *INPUT
¢go to 10                                                 *INPUT
¢end                                                      *INPUT
```

```
===> _
. ..+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
***** TOP OF FILE *****                                    /***/
00010PRINT 'ENTER THREE NUMBERS SEPARATED WITH COMMAS'     *===*
00020PRINT 'ENTER /* TO TERMINATE'                         *===*
00030L = I + J + K                                         *===*
00040M = I * &SQR2                                         *===*
00050N = J * &PI                                           *===*
00060PRINT I,J,K                                           *===*
00070PRINT L,M,N                                           *===*
00080PRINT 'END OF CALCULATION'                            *===*
00090GO TO 10                                              *===*
00100END                                                   *===*
***** END OF FILE *****                                    *****
```

Example 2:

This example does not show the LINEMODE command as such. It demonstrates, however, how you use linenumber editing once you have established line numbers as part of the member records (by using the LINEMODE command as shown in the preceding example).

First two lines are added behind line 70 by using the INSERT command. Then the 'nn' command is used to add a line behind line 80.

```
===> 70;prompt 3;i print ' ';i print ' ';u 2_
....+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
***** TOP OF FILE *****                                 /***/
00010PRINT 'ENTER THREE NUMBERS SEPARATED WITH COMMAS'  *===*
00020PRINT 'ENTER /* TO TERMINATE'                      *===*
00030L = I + J + K                                      *===*
00040M = I * &SQR2                                      *===*
00050N = J * &PI                                        *===*
00060PRINT I,J,K                                        *===*
00070PRINT L,M,N                                        *===*
00080PRINT 'END OF CALCULATION'                         *===*
00090GO TO 10                                           *===*
00100END                                                *===*
***** END OF FILE *****                                 *****
```

```
===> 85 print 'now we start again'_
00070PRINT L,M,N                                        /===/*
00073PRINT ' '                                          *===*
00076PRINT ' '                                          *===*
00080PRINT 'END OF CALCULATION'                         *===*
00090GO TO 10                                           *===*
00100END                                                *===*
***** END OF FILE *****                                 *****
```

```
===> _
00085PRINT 'NOW WE START AGAIN'                         /===/*
00090GO TO 10                                           *===*
00100END                                                *===*
***** END OF FILE *****                                 *****
```

LOCATE Command

LOCNOT Command

LOCUP Command

The LOCATE command scans the characters of each record (as defined by the column suffix or the ZONE setting) for the string specified.

```
Locate        string|/string/
LOCNot|LN
LOCUp|LUp
```

string is any group of characters to be searched for in the file. '/' is the string delimiter character (its specification is optional and only required if blanks must be included to the right). If 'string' includes double-byte characters and if the SO and/or SI control character(s) occupy the leftmost and/or rightmost position in it, they are not considered part of 'string'.

The column suffix Cnn can be used with this command.

The LOCATE command scans the characters of each record (as defined by the column suffix or ZONE setting) from the line following the current line through end-of-file for the specified character string. If an end of file condition immediately preceded the LOCATE, an automatic TOP request is performed before LOCATE begins. If 'string' is located, the pointer is positioned at the line that contains it. If 'string' is not located, the pointer is positioned to the last line of the file.

The command is not column-dependent. All characters are scanned, as specified by the ZONE command or the column suffix.

LOCATE can be used to scan for a line identifier in columns 73-80 assuming that the zone has been set to include these columns.

The LOCUP (locate up) version of the command causes a search backwards (or upwards) from the current line. If the line pointer happens to be at the top of the file, the LOCUP command has no effect.

When 'string' has been found, the line containing the specified string is displayed. If the string is not found, the *EOF message will be displayed.

The LOCNOT (locate not equal) version of the command proceeds forward through the file as does the LOCATE. However, the LOCNOT request is satisfied on the first unequal condition encountered within the zone rather than the first equal; that is, with the first line where the string does not occur in the current zone (see Example 4 below).

For column dependent scans see the FIND command and for scans through the entire file see the SEARCH command.

Examples:

Example 1:

Locate the string 'LOOPA'; search through all columns of each line.

```
===> locate loopa_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
***** TOP OF FILE *****                                     /***/
MAINPGM  CSECT                                              *===*
         PRINT GEN                                          *===*
         BALR  5,0                                          *===*
         USING *,5                                          *===*
         OPEN FILOUT                                        *===*
LOOPA    EXCP  RDCCB                                        *===*
         WAIT  RDCCB                                        *===*
         CLC   R(3),=C'/* '                                 *===*
         BE    ENDCARD                                      *===*
         MVC   0(80,2),R                                    *===*
         PUT   FILOUT                                       *===*
         B     LOOPA                                        *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
LOOPA    EXCP  RDCCB                                        /===/*
         WAIT  RDCCB                                        *===*
         CLC   R(3),=C'/* '                                 *===*
         BE    ENDCARD                                      *===*
         MVC   0(80,2),R                                    *===*
         PUT   FILOUT                                       *===*
         B     LOOPA                                        *===*
ENDCARD  CLOSE FILOUT                                       *===*
```

Example 2:

Locate the string 'LOOPA'; search through all columns starting in column 16.

```
===> locatec16 loopa_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
***** TOP OF FILE *****                                    /***/
MAINPGM  CSECT                                             *===*
         PRINT GEN                                         *===*
         BALR  5,0                                         *===*
         USING *,5                                         *===*
         OPEN FILOUT                                       *===*
LOOPA    EXCP  RDCCB                                       *===*
         WAIT  RDCCB                                       *===*
         CLC   R(3),=C'/* '                                *===*
         BE    ENDCARD                                     *===*
         MVC   0(80,2),R                                   *===*
         PUT   FILOUT                                      *===*
         B     LOOPA                                       *===*
ENDCARD  CLOSE FILOUT                                      *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
         B     LOOPA                                       /===/*
ENDCARD  CLOSE FILOUT                                      *===*
         OPEN  FILIN                                       *===*
LOOPB    GET   FILIN                                       *===*
```

Example 3:

Locate the string ' 59'; search upward from the current line toward the top of the file.

```
===> locup /  59/_
..<<+....1....+....2....+....3. .5....+.. MEM=$$PRINT  ...+..F>
  STMNT ERROR NO.   MESSAGE                                /===/*
                                  84-03-31                 *===*
    59   IPK156     SYMBOL 'PROCSAV4' UNDEFINED            *===*
                                                           *===*
```

```
===> _
..<<+....1....+....2....+....3. .5....+.. MEM=$$PRINT  ...+..F>
0000A0 0000 0000          59      ST    R13,PROCSAV4+4    /===/*
                     PUN07570                              *===*
         *** ERROR ***                                     *===*
0000A4 41D0 A1C0  0025A   60      LA    R13,PROSAV$4       *===*
                     PUN07580                              *===*
                          61      PUT   CONSOLO            *===*
                     PUN07590                              *===*
```

Example 4:

Find the first occurrence of a nonblank in position 15.

```
===> zone 15 15;locnot / /_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
MAINPGM  CSECT                                             /===/*
         PRINT GEN                                         *===*
         BALR  5,0                                         *===*
         USING *,5                                         *===*
         OPEN  FILOUT                                      *===*
```

```
===> _
....+....1...>><...2....+....3. .5....+.. MEM=SAMPASMB ...+..FS
              DEVICE=3340,EOFADDR=ENDDISK                  /===/*
         ORG   FILIN+22                                    *===*
         DC    C'IJSYS01'                                  *===*
         ORG                                               *===*
FILOUT   DTFSD IOAREA1=IOA,IOAREA2=IOB,IOREG=(2),        X *===*
               TYPEFLE=OUTPUT,BLKSIZE=408,RECSIZE=80,    X *===*
               DEVICE=3340,RECFORM=FIXBLK                  *===*
```

Example 5:

Find the first occurrence of the two double-byte characters specified in 'string'.

```
*************** SamPle 5.Part 1 ********************

===> L /SO小倉SI/
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                             /***/
ADDRESS1 = SOG '東京都港区六本木３丁目' SI;                              *===*
 ADDRESS2 = SOG '東京都新宿区西新宿２丁目' SI;                           *===*
  ADDRESS3 = SOG '広島県広島市稲荷町４丁目' SI;                          *===*
   ADDRESS4 = SOG '福岡県北九州市小倉北区紺屋町' SI;                     *===*
    ADDRESS5 = SOG '東京都新宿区新宿７丁目' SI;                          *===*
     ADDRESS6 = SOG '千葉県松戸市新松戸７丁目' SI;                       *===*
      ADDRESS7 = SOG '東京都千代田区永田町１丁目' SI;                    *===*
       ADDRESS8 = SOG '神奈川県藤沢市桐原町１' SI;                       *===*
***** END OF FILE *****                                             *****
```

```
*************** SamPle 5.Part 2 ********************

===>
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
   ADDRESS4 = SOG '福岡県北九州市小倉北区紺屋町' SI;                     /===/*
    ADDRESS5 = SOG '東京都新宿区新宿７丁目' SI;                          *===*
     ADDRESS6 = SOG '千葉県松戸市新松戸７丁目' SI;                       *===*
      ADDRESS7 = SOG '東京都千代田区永田町１丁目' SI;                    *===*
       ADDRESS8 = SOG '神奈川県藤沢市桐原町１' SI;                       *===*
***** END OF FILE *****                                             *****
```

@MOVE Editor Macro (see @COPY Editor Macro)

MSG Command (see /MSG System Command)

NEXT Command

The NEXT command advances the line pointer in the file by 'n' lines.

```
Next          [n|1]
DOwn
```

n indicates the number of lines by which the pointer is to be advanced. The maximum number allowed is 99999. If n is unspecified, 1 is assumed. If end-of-file is reached before the line pointer is advanced n lines, the pointer is positioned at the last line, and the INVALID RANGE message appears.

When verification is on, the new current line is displayed. To restore the line pointer to the top of the file, issue the TOP command.

Notes:

1. *If an index has been built using the INDEX command and the number of lines to be advanced is large (greater than 100 or your specified index value), it will take less time to advance the pointer by using the POINT command. Or, if you were using linenumber editing when you built the index, the pointer can be rapidly repositioned by specifying the sequence number desired.*

2. *In edit mode, pressing the ENTER key with no data input is assumed to be a request to move the line pointer to the next line. In other words, it has the same effect as 'N 1'.*

OVERLAY Command

The OVERLAY command is used to overlay the current line with the characters specified in the operand field of the command.

```
Overlay      string
OVERLAYX
OX
```

string is a string that replaces parts of the current line.

The column suffix Cnn can be used with this command.

Blank characters make no change to the characters in the corresponding positions of the current line. If there is more than one space after the request, these spaces are considered to be part of 'string'.

If you want to blank out a character in the record being edited, use the delimiter character (usually /) in the string. It will overlay the corresponding position with a blank character.

The OVERLAYX and OX (overlay hex) version of this command lets you insert hexadecimal character combinations into the source line being edited. Instead of typing one overlay character per column to be replaced, type in two hexadecimal digits for each column to be replaced. If invalid or unpaired hexadecimal digits are encountered, the digits are assumed to be valid character data so that both character and hex data can be inserted using the same command.

The REPEAT command can be entered before the OVERLAY command to extend the effect of the command to more than a single line.

If the *EOF message is displayed, end of file was reached by the request. For OVERLAY to reach the end of file, the REPEAT request had to be entered before the OVERLAY.

Examples:

Example 1:

```
===> repeat 9;overlayc71 **;top_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                     /***/
***************************** **********************        *===*
** THIS                                                     *===*
**     PROGRAM                                              *===*
**             PROLOG                                       *===*
**                     SHOULD                               *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                     /***/
****************************** ********************** *===*
** THIS                                           ** *===*
**     PROGRAM                                    ** *===*
**             PROLOG                             ** *===*
**                     SHOULD                     ** *===*
**                            HAVE                ** *===*
**                                 A NEAT         ** *===*
**                                        FRAME   ** *===*
****************************** ********************** *===*
***** END OF FILE *****                              *****
```

Example 2:

Insert a REP statement in an object deck. The REP statement needs a X'02' in position 1. The 'OVERLAYX 02' command replaces the '$' which was placed in position 1 via the INSERT command.

```
===> l /rld /;insert $rep  0000e8 00147f0;overlayx 02_
<<..+....1....+....2....+....3. .5....+.. MEM=TPROGOBJ>>..+..FS
***** TOP OF FILE *****                                     /***/
CATALOG TPROG.OBJ         EOD=/+          REPLACE=YES       *===*
 ESD             TPROG          IJ2L0010                    *===*
 TXT               K  Ys K    h&       0s         s  1      *===*
 TXT               30        K     .    0 ¢       0         *===*
 TXT                                                        *===*
 TXT                                       &                *===*
 TXT                                                        *===*
 TXT    y        HERE IS THE TEST PROGRAM                   *===*
 RLD                                               D        *===*
 END                                                        *===*
/+                                                          *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=TPROGOBJ>>..+..FS
 REP  0000E8 00147F0                                        /===/*
 END                                                         *===*
/+                                                           *===*
/*                                                           *===*
***** END OF FILE *****                                      *****
```

PFnn Command

The PFnn command simulates the function of a Program Function key. It is normally used when no PF keys are available.

```
PFnn
```

nn is a decimal number from 1 to 24 indicating the PF key function requested.

Entering a PFnn command has the same effect as pressing PF key 'nn'. You can place the PFnn command in a list of commands separated from one another by logical line-end characters. The command to which the PFnn command is equated will only be executed after all the other commands in the list.

If multiple PFnn commands are entered in a list, or in separate command areas, only the function associated with the last PFnn command on the physical screen will be performed. If a PFnn command is entered on the screen and an actual PF key is pressed, the PFnn command will override the PF key.

A PF key (or PFnn command) within the full screen editor can be equated to multiple commands separated by logical line end characters. When the PF key is pressed, or the PFnn command is entered, the commands associated with the PF key are performed after all other command activity on the screen.

The invoked function is selected from the editor PF key set 'PFED', if it is defined and active. Otherwise, if this set is not defined, or is cleared, or is set off, the function is selected from the command mode PF key set 'PF'.

The /PFnn system command can also be used. This command is restricted to its normal use outside the full screen editor and thus, its effect differs from that of the PFnn command. Specifically, the /PFnn command can only invoke a single command or data line; if it is encountered in a command list, it is processed as it occurs in the list rather than at the end like the PFnn command.

POINT Command

The POINT command positions the line pointer to a given area of the file based on an index constructed with the INDEX command (see page 4-86).

```
POint         nn|Snn|Pnn
```

nn is a decimal number from 1 to 9999 which represents the relative (to 1) line number within the file to which the line pointer is to be set.

Snn is a decimal number consisting of up to 8 digits preceded by an 'S' which represents the sequence number of the line to which the line pointer is to be set.

Pnn is a decimal number from 1 to 32 preceded by a 'P' indicating that the line pointer is to be set to the first line of an index table page. The number of lines in a page is determined by the operand specified when the INDEX command was issued.

When the 'nn' operand is specified, the line pointer is set to approximately the 'nn'th line within the file. The approximation becomes less exact if many additions and/or deletions have taken place in this area of the file after the INDEX command.

When the 'Snn' operand is specified, the line pointer is set to the line on which the specified sequence number occurs. If the sequence number is not found, the line pointer will be set to the line with the next lower sequence number. This form of the POINT command requires that linenumber editing be in effect from the time when the INDEX command is entered. See also the 'nn' command on page 4-140 for an easier way of using the POINT Snn function.

When the 'Pnn' operand is specified, the line pointer will be positioned to the line associated with the 'nn'th entry within the index. For example, if the INDEX was built with an operand of 100, P1 would point you to the 1st line in the file, P2 to the 100th, P3 to the 200th, etc. This form of the command can be used to initially determine where the index pointers are within the file being edited.

If a line representing one of the index points within the file is deleted, the corresponding entry is also deleted from the index thus causing, perhaps, slower responses when pointing to lines immediately beyond the affected point in the file. In this situation you could reenter the INDEX command to build a new index.

Using the POINT command to move the line pointer long distances can slow response time (the INDEX command should be used to speed the movement of the line pointer).

Examples:

1. `point 1200`

   ```
   The line pointer will be set to approximately the
   1200th line in the file.
   ```

2. `point S198500`

   ```
   The line pointer will be set to the line containing
   sequence number 198500.
   ```

PRINT Command

The PRINT command displays lines from the member currently being edited or from the print, punch (stack) and log areas. The data to be displayed can be alphanumeric or mixed data.

```
Print          [HEX] [n1|* [n2|*|72]]
TYpe                 $$LOG
PRINTFwd             $$STACK
PF                   $$PRINT
               [S]
```

HEX	specifies that lines will be displayed in both character and hexadecimal format.
n1	is the number of lines to be displayed. The maximum value is 99999. If '*' is specified, the remainder of the file up to a maximum of 99999 lines is displayed. The default is one line for a typewriter, and one screen for a display terminal.
n2	specifies the last column of each line to be printed. The default is the line size value at the time the editor is entered (usually 72). If a width value has been set in the VERIFY command, this value will be the default print width value. If '*' is specified, all 80 columns will be displayed. 'n1' is required if 'n2' is to be specified.
$$LOG	displays your log area.
$$STACK	displays the editor stack (which is identical to the punch area).
$$PRINT	displays the last print output (in the print area) from the last execution.
S	causes a scale line to be printed, which is useful if you are entering fixed format data.

The PRINT command displays n1 lines, starting with the current line. If n2 is not specified, the default value is taken. 'n2' can be between 1 and 80. The line pointer is not changed by the PRINT command.

If PRINTFWD or PF is specified, the line pointer is advanced as the display proceeds. This is useful for scrolling through files.

If more lines are requested than fit on one screen, pressing ENTER would give you the next screen. As an alternative, you may place the terminal in continuous output mode with the /CONTINU command. This will display the remaining output automatically.

The /SKIP command can be used to forward or backward space the display if the total lines exceed one screen.

The /CANCEL command (or 3270 PA2 key) can be used to terminate a display in progress. Your terminal then returns from list to edit mode.

Note: The PRINT command displays data in 80-character record format.

Examples:

Example 1:

Display the contents of an object deck in both character and hexadecimal formats.

```
===> print hex_
<<..+....1....+....2....+....3. .5....+.. MEM=TPROGOBJ>>..+..FS
***** TOP OF FILE *****                                   /***/
 ESD           TPROG            IJ2M0011                  *===*
 TXT             K  Ys K    h&         &  D       s  1    *===*
 TXT                  Dq        $ s    u       s g    n   *===*
 TXT    Y                 s                               *===*
 TXT                  &         &                         *===*
 TXT             30      00 k      00 ¢           j     0 *===*
```

```
 ...+....1....+....2....+....3. .5....+....6....+....7....+..LS
  ESD            TPROG           IJ2M0011
 0CEC444444034400EDDDC4440007400 CDFDFFFF0000400344444444
 2524000000000001379670000008002 9124001100000000800000000
  TXT                K  Ys K    h&          &  D      s  1
 0EEE4007440344000AD1AEA2D3A0A85 009ED01A5DAC4DAC51A25F01
 2373000800080001502708 2E2710180 AE0A0C8F00141010802A8100
  TXT                     Dq     $ s    u      s g    n
 0EEE400B440344004E005DAC9ED00F9 50A042AA111250A082A29420
 2373000000080001 5F0C80148A0C7E0 B02A100482A0802E700E5000
  TXT   Y                      s
 0EEE400E4403440048A212420051A05 0080000000002000000000000
 2373000800080001 7002B1120180268 0000000400100000000003000
  TXT                     &
 0EEE4002440144000000B00050000000 4444444444444444444444444
 2373001000020001901200000000000 0000000000000000000000000
  TXT                30      00 k      00 ¢           j     0
 0EEE4000440344000 3FF03004FF3901 124FF242124211009810 41F3
 2373000000080002A230A200700C290 08700EA008000EA01002700A
  TXT                  K     k    0  00;       .
 0EEE4003440344000 00FD0111290110 F44FF5421242100F44444444
 2373000800000002A77E210E082A08A 06700E8008B0007E00000000
  TXT                                  &          &
```

Example 2:

A typical use of the PRINT command under the full screen editor is the displaying of the $$LOG area. This example shows a somewhat unusual application of the COPY macro.

Assume you modified the COPY macro as indicated by the arrows below:

```
        ===> _
        <<..+....1....+....2....+....3. .5....+.. MEM=COPY    >>..+..FS
        ***** TOP OF FILE *****                                  /***/
        @MACRO    COPY (# LINES,DIRECTION,DISPLACEMENT)          *===*
        @NOPR     E.G. @COPY 3 DOWN 5  OR  @COPY 2 LOC  XYZ      *===*
        STACK OPEN                                               *===*
==>     SET LOG ON INOUT                                         *===*
==>     SET LOG ON INOUT ------------------------                *===*
        STACK &&PARAM1                                           *===*
        STACK CLOSE                                              *===*
        &&PARAM2 &&PARAM3                                        *===*
==>     SET LOG OFF                                              *===*
        GETFILE $$STACK                                          *===*
==>     PRINT $$LOG                                              *===*
        ***** END OF FILE *****                                  *****
```

Now open an editing session for member TEST2. You want to copy the portion from top-of-file down to the first line that contains 'ENDFROM' to the place where 'BEGINTO' occurs first.

You had inserted the command 'PRINT $$LOG' in the COPY macro. This produces the display of the $$LOG area as shown in the second screen below (it must be read from bottom to top).

The arrows in the last part of the figure point to important error messages. As you can see from the message 'STACK IS FULL', the copy operation did not complete successfully; the string 'ENDFROM' was not found within the first 99 lines. Also, the target location was not found (message '*EOF').

```
===> @copy /endfrom/ locate /beginto/
<<..+....1....+....2....+....3. .5....+.. MEM=TEST2   >>..+..FS
***** TOP OF FILE *****                                  /***/
RECORD    00001                                          *===*
RECORD    00002                                          *===*
RECORD    00003                                          *===*
RECORD    00004                                          *===*
RECORD    00005                                          *===*
```

```
         _
         ...+....1....+....2....+....3. .5....+....6....+....7....+..LS
         I 15/28 ED   SET LOG OFF
  ==>    O 15/28      *EOF
         I 15/28 ED   LOCATE   /BEGINTO/
         I 15/28 ED   STACK CLOSE
  ==>    O 15/28      STACK IS FULL
         I 15/28 ED   STACK /ENDFROM/
         O 15/28      *CONTROL SET
         I 15/28 ED   SET LOG ON INOUT ------------------------
         O 15/28      *CONTROL SET
         I 15/20 ED   SET LOG OFF
```

You must press PA2 (cancel) and ENTER to return to your edit session.

PROMPT Command

The PROMPT command is used to set the current prompt increment, and to change this increment from its default of 10 to a value that you specify.

```
PROmpt          [nn|10]
```

nn is a decimal number from 1 to 32767 indicating the desired prompt increment.

The prompt increment is used during linenumber editing for forming sequence numbers for lines entered via the INPUT command.

A PROMPT command with no operands causes the current prompt increment to be displayed.

Example:

Insert two lines behind line 70 with a prompt increment of 3.

```
===> 70;prompt 3;I print ' ';i print ' ';u 2_
....+<<..1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
***** TOP OF FILE *****                                  /***/
00010PRINT 'ENTEP THREE NUMBERS SEPARATED WITH COMMAS'   *===*
00020PRINT 'ENTER /* TO TERMINATE'                       *===*
00030L = I + J + K                                       *===*
00040M = I * &SQR2                                       *===*
00050N = J * &PI                                         *===*
00060PRINT I,J,K                                         *===*
00070PRINT L,M,N                                         *===*
00080PRINT 'END OF CALCULATION'                          *===*
00090GO TO 10                                            *===*
00100END                                                 *===*
***** END OF FILE *****                                  *****
```

```
===> _
00070PRINT L,M,N                                         /===/*
00073PRINT ' '                                           *===*
00076PRINT ' '                                           *===*
00080PRINT 'END OF CALCULATION'                          *===*
00090GO TO 10                                            *===*
00100END                                                 *===*
***** END OF FILE *****                                  *****
```

QUIT Command

The QUIT command terminates edit processing for the file associated with the logical screen on whose command line it is entered.

```
Quit
END
```

All storage areas associated with the file being edited are released.

If there is only one file being edited (that is, all previous files accessed via the ENTER command have been QUIT), the QUIT command terminates the full screen editor and returns you to command mode.

If more than one file is being edited, the QUIT command terminates editing of the file indicated. The logical screen associated with the file being terminated will be released to another file.

If the QUIT command is used for a library member, the changes made to the member will not be affected, since these have been applied directly to the member.

If you enter the QUIT command for a newly created file, the file will be lost, unless you had previously saved it with the SAVE command. However, you should not use QUIT and SAVE to save newly created members; it is better to use the FILE command.

If the QUIT command is used for the file associated with the input area, the contents of the input area are not saved. However, these contents will still be available after the full screen editor has been terminated, either for editing or saving.

Note: Be careful when following the QUIT or FILE command with other editor commands separated with logical line-end characters; the commands will be applied to a different file.

RENUM command

The RENUM command is used to resequence or renumber the file being edited.

```
RENum        [incr|100 [scol|73[n|8 [strt|incr]]]]
```

incr is the increment by which the file is resequenced. The default is 100, unless you are using linenumber editing. In this case, the default is ten times the prompt, or 100, whichever is greater. The maximum increment is 9999.

scol is the starting column number for the sequence number field. The default is column 73 unless linenumber editing is in effect, in which case the 'linemode' sequence number location is used. When you are editing mixed data, only 1 and 73 are valid starting columns.

n is the number of columns in the sequence number field. The default is 8 unless you are using linenumber editing, in which case the line number location and size are used as defaults. The maximum number of columns is 16. If you resequence a DBCS member and if scol has a value of 73, n must be 8.

strt is the starting sequence number. If omitted, the starting sequence number is assumed to be the same as the sequencing increment. The maximum value that can be specified is 9999.

The operands control the location and size of the sequence number and the increment.

If you are using linenumber editing, you need not specify the location of the sequence number field; the line number sequence number location will be used for the renumbering. In this case, the increment used will be 100, or ten times the current prompt increment, whichever is greater. However, any resequencing increment may be explicitly specified even if linenumber editing is in effect.

The line pointer is not altered by the renumbering operation.

If an index is in effect when the RENUM command is issued, its effect will be negated and the INDEX command must be reissued (see INDEX command).

Note: If the sequence number starts below column 73, statements with a '/' in column 1 will not be resequenced.

Example:

```
===> linemode left;renum 10;top_
<<..+....1....+....2....+....3. .5....+.. MEM=CALC2   >>..+..FS
***** TOP OF FILE *****                                  /***/
00010PRINT 'ENTER THREE NUMBERS SEPARATED WITH COMMAS'   *===*
00020PRINT 'ENTER /* TO TERMINATE'                       *===*
00030L = I + J + K                                       *===*
00040M = I * &SQR2                                       *===*
00043N = J * &PI                                         *===*
00046PRINT I,J,K                                         *===*
00050PRINT L,M,N                                         *===*
00060PRINT 'END OF CALCULATION'                          *===*
00070GO TO 10                                            *===*
00080END                                                 *===*
```

```
===>
....+<<..1....+....2....+....3. .5....+.. MEM=CALC2    ...+..F>
***** TOP OF FILE *****                                  /***/
00010PRINT 'ENTER THREE NUMBERS SEPARATED WITH COMMAS'   *===*
00020PRINT 'ENTER /* TO TERMINATE'                       *===*
00030L = I + J + K                                       *===*
00040M = I * &SQR2                                       *===*
00050N = J * &PI                                         *===*
00060PRINT I,J,K                                         *===*
00070PRINT L,M,N                                         *===*
00080PRINT 'END OF CALCULATION'                          *===*
00090GO TO 10                                            *===*
00100END                                                 *===*
```

REPEAT Command

The REPEAT command executes the ALIGN, BLANK, CENTER, JUSTIFY, OVERLAY or SHIFT command 'n' times.

```
REPEAT        [n|*|1]
RPT
```

n is an integer specifying the number of times that the command will be repeated. If '*' is specified the operation will proceed from the line pointer to end-of-file, or to the maximum number of repetitions (9999).

If 'n' is not specified, it is assumed to be 1. If 'n' is greater than the number of lines between the current line and end-of-file, REPEAT remains in effect until the end of the file.

REPEAT remains in effect until the edit session is ended, or until you enter an ALIGN, BLANK, CENTER, JUSTIFY, OVERLAY or SHIFT command. The repeat value is then reset to 1.

Example:

```
===> set num on;zone 1 80;repeat *;blankc73 ————————;top_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
***** TOP OF FILE *****                                  /***/
Unfortunately, this text member has been entered         000100
with 'LINEMODE RIGHT'; of course, the sequence           000200
numbers in column 73—80 must be removed. This is         000300
a good application for the REPEAT command.               000400
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA* ...+..F>
***** TOP OF FILE *****                                  /***/
Unfortunately, this text member has been entered
with 'LINEMODE RIGHT'; of course, the sequence
numbers in column 73—80 must be removed. This is
a good application for the REPEAT command.
```

REPLACE Command

The REPLACE command replaces an existing library member with the contents of the input area or with a newly created file.

```
| REPlace    [name [password] [PRIV|PUBL] [DBCS=ON|OFF]]
```

name is the name of the library member to be replaced.

password is a 4-character word and need only be specified if the member being replaced is password protected.

PRIV|PUBL can be specified if you want to assign the private or public attribute to the replacement member. If neither PRIV nor PUBL is specified, the previous privacy attribute of the member is retained.

| DBCS=ON|OFF allows you to set or reset the DBCS attribute for the new member. If the DBCS operand is not specified, the member will have the attribute that was set during the editor session by a SET DBCS command. If a SET DBCS command has not been specified either, the attribute of the old member is retained.

The 'name' operand specifies the name of an existing library member. If REPLACE is entered for the newly created file (see ENTER command), 'name' will override the name specified in the ENTER command. If 'name' is not specified, the 'name' is taken from the ENTER command. The characters '.P' at the end of the member name indicate a print-type member.

Example:

```
ed                     (Edit the input area)
get membrb  1 50       (Put some records in it)
replace membra         (Replace MEMBRA with the contents of the
                        input area)
```

RESTORE Command

The RESTORE command restores the editor settings that were in effect when the previous STACK EDIT command was issued.

```
RESTore
```

Several levels of STACK EDIT and RESTORE can be nested so that settings can be saved, changed, saved, changed, etc. and then restored, restored, restored, etc.

The settings for the following editor commands are restored:

```
CASE     LINEMODE     VERIFY
FLAG     DELIM        ZONE
IMAGE    TABSET       SET (for control characters only)
```

RIGHT Command

The RIGHT command shifts the displayed data to the right within a logical screen.

```
RIght         [nn|1]
```

nn is a decimal number indicating the number of columns by which the data is to be shifted to the right.

It is often necessary to view columns of data which are not displayed on the screen. There are two ways to do this. One is by using the VIEW command to display only certain columns (see the VIEW command). The second method is to shift the existing data right (or left) using the RIGHT (or LEFT) command.

When you issue the RIGHT command, the data within the logical screen will be shifted 'nn' columns to the right. If 'nn' is omitted, 1 column is assumed. No data loss occurs and you have only to enter the LEFT command to cause the data to be returned to its previous position.

The number of columns shifted to the left from the normal or base position is called the offset. If a non-zero offset exists, it will be displayed in the scale/header line as a reminder. A maximum offset of 149 is accepted. To return the display to the normal or base position, issue a LEFT command for as many columns as are indicated in the offset.

When you shift mixed data and when a double-byte character would occupy the rightmost position on the screen after the shift operation, this double-byte character will not be displayed. It is replaced by one or two control characters, which guarantee that the remainder of the subfield is displayed correctly. The data in the VSE/ICCF library is not affected by this operation, as long as you do not:

- Replace the added control characters
- Change double-byte characters to one-byte characters.

When you shift mixed data, and an SO control character would occupy the rightmost position on the screen after the shift operation, the control character will be displayed as an unprintable character.

SAVE Command

The SAVE command is used to save a newly created file, or to save the input area as a new library member.

```
SAve          ┌                         ┌    ┐┐  [DBCS=ON|OFF]
              │name [password]          │PRIV││
              │                         │PUBL││
              └                         └    ┘┘
```

name	is a 1 to 8 character word beginning with an alphabetic character which is to be the name for a newly created file, or for the input area when it is being saved in the library. For a newly created file this name will override the name specified on the ENTER command. The characters '.P' at the end of the member name indicate a print-type member.
password	is a 4 character word which can be specified to password protect a newly created file being saved in the library.
PRIV|PUBL	allows you to specify that a newly created member is to be saved in the library as private or public. If neither operand is specified, the private or public status will be set according to the default in your user profile.
DBCS=ON|OFF	allows you to save the new member with the DBCS attribute. If the DBCS operand is not specified, the new member is saved with the attribute that was set during the editor session by a SET DBCS command. If a SET DBCS command has not been specified either, DBCS=OFF is assumed for the new member.

The SAVE command when used for a newly created file (that is, a file which is not already a member of the library-- see the ENTER command) or when issued for the input area file, causes the file to be saved in the library. After the SAVE command is issued, editing can continue on the file; however, instead of editing a newly created file or the input area, you will now be editing a library member.

If the SAVE command is used for a newly created file, no 'name' operand is required. In this case the file is saved in the library under the name specified in the ENTER command. If a 'name' operand is specified, the operand overrides the name specified on the ENTER command.

If the SAVE command is used for the input area, the 'name' operand must be specified and the input area will be saved in the library under this name.

If the SAVE command is used for a file which is already in the library, the 'name' operand must not be specified and a 'save' operation will not be performed since all changes to the member have already been applied directly to the member. However, the SAVE command will cause any buffers retained in storage by VSE/ICCF to be written to disk, thus protecting you against data loss should a system failure occur.

Example:

```
ed                        (Edit the input area)
get membrb 1 50           (Put some records in it)
save newfil               (Save it in the library)
top                       (Continue editing NEWFIL)
```

SCREEN Command

The SCREEN command is used to specify the number and size of logical screens within the physical screen. It also displays the current setting of logical screens.

```
SCReen        [nn [nn [nn ...]]]
```

nn ...are decimal numbers representing the number of physical lines within the corresponding logical screen.

Each operand represents one logical screen. The operand itself represents the number of physical lines within the logical screen. Up to 8 operands can be specified representing a maximum of 8 logical screens.

If the total number of lines specified in all logical screens is less than the actual physical screen size, the remaining lines will be assigned to the final logical screen if the number of remaining lines meets a minimum requirement.

If the SCREEN command is entered with no operands, the current setting of the SCREEN command will be displayed in the first command area associated with the file being edited.

The minimum number of lines in a given logical screen depends upon the number of physical lines associated with the command area. A minimum logical screen must contain two lines, one scale/header line, and one command area. The command area can be from 1 to 4 physical lines as determined by the third operand of the VERIFY command.

The full screen editor controls its own screen formatting. Thus, all 'SET' or '/SET SCREEN' command specifications are ignored while the full screen editor is in control.

If the number of logical screens defined by the SCREEN command is greater than or equal to the previous number of logical screens, the files displayed on the screen will remain on the screen. However, if the screen sizes themselves are smaller than the previous screen sizes, the format specifications (see FORMAT command) can be adjusted so that the format areas fit within the logical screen. (Format areas will automatically be reduced to fit within a smaller logical screen but will not automatically be expanded to encompass a larger logical screen.)

If the number of logical screens defined by the SCREEN command is greater than the previous number of logical screens, the extra new screens will be assigned when the ENTER command is issued for files not displayed on the screen.

If the number of logical screens defined by the SCREEN command is less than the previous number of logical screens, the first available screen will be assigned to the file associated with the logical screen where the SCREEN command was entered. Any subsequent screens will be assigned to files not on the screen, beginning with the oldest file.

Note: *When changing SCREEN specifications, other editor commands should not be specified on the same physical screen. Because commands are processed from the top to the bottom of the screen, any editor commands following the SCREEN command itself will be based on the new setting of the SCREEN parameters. Thus, when changing SCREEN specifications, the SCREEN command should be the only command entered prior to pressing ENTER.*

Example:

Split the screen into two logical screens. Use the lower screen to edit MEMBER2. After having completed editing MEMBER2, go back to a display of one logical screen only (by issuing 'screen 24').

```
===> screen 12 12;enter member2_
<<..+....1....+....2....+....3.  .5....+.. MEM=MEMBER1 >>..+..FS
***** TOP OF FILE *****                                     /***/
RECORD  1  OF MEMBER1                                       *===*
RECORD  2  OF MEMBER1                                       *===*
RECORD  3  OF MEMBER1                                       *===*
RECORD  4  OF MEMBER1                                       *===*
RECORD  5  OF MEMBER1                                       *===*
RECORD  6  OF MEMBER1                                       *===*
RECORD  7  OF MEMBER1                                       *===*
RECORD  8  OF MEMBER1                                       *===*
RECORD  9  OF MEMBER1                                       *===*
```

```
===>
<<..+....1....+....2....+....3.  .5....+.. MEM=MEMBER1 >>..+..FS
***** TOP OF FILE *****                                     /***/
RECORD  1  OF MEMBER1                                       *===*
RECORD  2  OF MEMBER1                                       *===*
RECORD  3  OF MEMBER1                                       *===*
RECORD  4  OF MEMBER1                                       *===*
RECORD  5  OF MEMBER1                                       *===*
RECORD  6  OF MEMBER1                                       *===*
RECORD  7  OF MEMBER1                                       *===*
RECORD  8  OF MEMBER1                                       *===*
RECORD  9  OF MEMBER1                                       *===*
===> quit;screen 24_
<<..+....1....+....2....+....3.  .5....+.. MEM=MEMBER2 >>..+..FS
***** TOP OF FILE *****                                     /***/
RECORD NUMBER  1 OF MEMBER2                                 *===*
RECORD NUMBER  2 OF MEMBER2                                 *===*
RECORD NUMBER  3 OF MEMBER2                                 *===*
```

```
===> _
<<..+....1....+....2....+....3.  .5....+.. MEM=MEMBER1 >>..+..FS
***** TOP OF FILE *****                                     /***/
RECORD  1  OF MEMBER1                                       *===*
RECORD  2  OF MEMBER1                                       *===*
RECORD  3  OF MEMBER1                                       *===*
RECORD  4  OF MEMBER1                                       *===*
RECORD  5  OF MEMBER1                                       *===*
RECORD  6  OF MEMBER1                                       *===*
RECORD  7  OF MEMBER1                                       *===*
RECORD  8  OF MEMBER1                                       *===*
RECORD  9  OF MEMBER1                                       *===*
RECORD 10  OF MEMBER1                                       *===*
RECORD 11  OF MEMBER1                                       *===*
```

SEARCH Command

The SEARCH command scans the characters of each line (as defined by the column suffix or ZONE setting) from the beginning of the file through the end-of-file for the specified character string.

```
Search          string|/string/
```

| string is any group of characters to be searched for in the file. If 'string' includes double-byte characters and if the SI and/or SI control character(s) occupy the leftmost and/or rightmost position in it, they are not considered part of 'string'.

/ is the string delimiter character and is required when blanks must be included on the right.

The column suffix can be used with this command.

This command acts exactly like a TOP command followed by a LOCATE command. Whereas the LOCATE command searches the file from the current pointer location through the end-of-file, the SEARCH command always starts at the top of the file and, thus, searches the entire file.

If the string is located, the pointer is positioned at the line that contains it. If string is not located, the pointer is positioned at the last line of the file; the message *EOF will be displayed.

The request is not column-dependent because all characters are scanned as specified by the ZONE request or the column suffix.

For column dependent scans, see the FIND command. For current line to end-of-file scans, see the LOCATE command.

Examples:

Example 1:

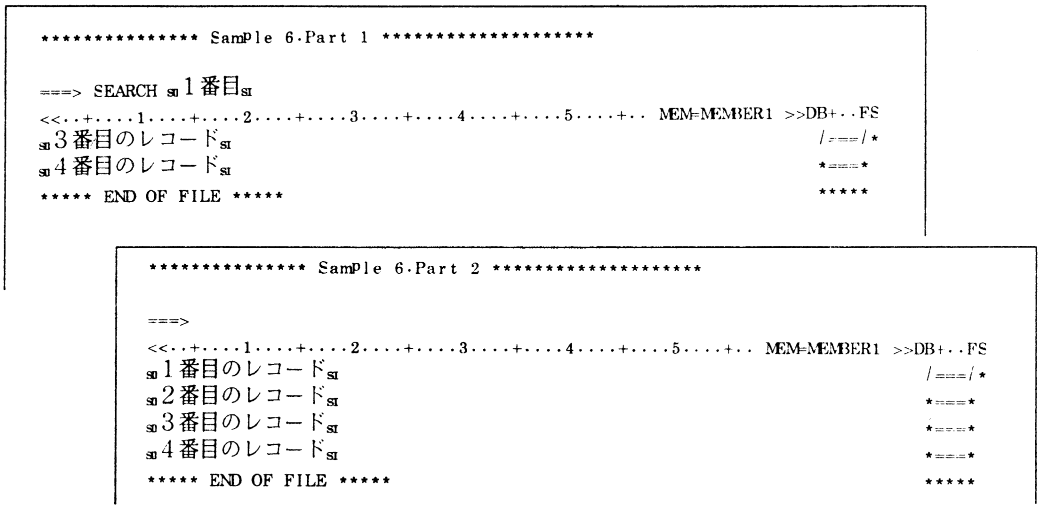
Find the string 'FIRST' with the search beginning at the top of the file.

```
===> search first_
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
THIRD RECORD                                                /===/*
FOURTH RECORD                                               *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. INP=*INPARA*>>..+..FS
FIRST RECORD                                                /===/*
SECOND RECORD                                               *===*
THIRD RECORD                                                *===*
FOURTH RECORD                                               *===*
***** END OF FILE *****                                     *****
```

Example 2:

Locate the three double-byte characters which are specified in the search argument; start the search at the top of the file.



```
*************** Sample 6-Part 1 ********************

===> SEARCH SO1番目SI
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
SO3番目のレコードSI                                                  /===/*
SO4番目のレコードSI                                                  *===*
***** END OF FILE *****                                             *****
```

```
*************** Sample 6-Part 2 ********************

===>
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
SO1番目のレコードSI                                                  /===/*
SO2番目のレコードSI                                                  *===*
SO3番目のレコードSI                                                  *===*
SO4番目のレコードSI                                                  *===*
***** END OF FILE *****                                             *****
```

SET Command

The SET command is used to set various VSE/ICCF functions on or off. The majority of these options is described under the /SET system command; see "[/]SET Command" on page 3-111. The following options are available only under the full screen editor.

```
SET          DBCS        ┌    ┐
             NULls       │ON  │
             NUMbers     │OFF │
             PFC         │    │
             REPoption   └    ┘

             DBCS=ON|OFF
```

ON sets the function on. This is the default.

OFF sets the function off.

Each option applies to the file being edited. Thus, you can have different option settings for different files. If neither ON nor OFF is specified in the SET command, ON is assumed in all cases.

The DBCS operand allows you to set or reset the DBCS attribute for the member that is being edited.

The NULLS option controls whether or not trailing blanks within a record are replaced with 'null' characters before it is transmitted to the screen. The NULLS option is automatically set ON when the full screen editor is first entered, or when a file is entered for the first time.

The NULLS option is useful in that it allows data to be added to a record on the screen using the 'insert mode' feature of the 3270 display terminal. Also, on remote terminals, use of the NULLS option can reduce transmission time. When using the NULLS option, be sure always to use the space bar rather than the Right Arrow key for all forward spacing of the cursor since 'null' characters will not be transmitted back to the processor, whereas 'space' characters will.

When the NULLS option is set off, the trailing blanks will be transmitted to the screen. Thus, it would be impossible to use the 'insert mode' feature without first converting trailing blanks to 'nulls' within the record by using the ERASE EOF key or by using the DEL key to add new nulls to the end of the record. The NULLS OFF option is useful when using the CURSOR command for changes beyond the last non-blank data within the record.

When the NUMBERS option is set on, sequence numbers are displayed within the line command area, instead of the normal '*===*' identifier. This option is especially useful when editing files which have sequence numbers in columns 73 through 80. For these files you need not sacrifice your line command area to see your sequence numbers.

If linenumber editing is in effect, the sequence number displayed in the line command area will be the rightmost six digits of the line number. If linenumber editing is not in effect, the data from columns 75 through 80 will be displayed if column 80 is numeric or, if not numeric, the data from columns 74 through 79 will be displayed if column 79 is numeric. Otherwise, the data from columns 73 through 78 will be displayed in the line command area.

When the NUMBERS option is in effect, line commands must begin in column 1 of the line command area and must be followed by a blank (space).

The NUMBERS option is normally off when the full screen editor is entered, or when a file is later entered via the ENTER command for the first time.

The PFC option controls the disposition of commands or lines of data associated with PF keys. This option is normally on unless specifically set off. When it is on, the data associated with a Program Function key is assumed to represent a command. It is processed as if the command had been entered in the command area associated with the current cursor position. The cursor does not have to be in the command area when you press the PF key.

When the PFC option is set off, the data associated with the Program Function key (except for CURSOR commands) is interpreted according to the position of the cursor. If the cursor is in the command area, the processing of the Program Function key is the same as if the PFC option were on. However, if the cursor is placed within a line, the data associated with the PF key replaces the line.

The REPOPTION option controls the interpretation of modified lines from the screen. The REPOPTION option is normally set off unless specifically set on. When the REPOPTION is off and a line on the screen is modified, if the line contains no logical tab characters (or logical tabbing is not in effect), the line is interpreted as a modification to an existing line and is interpreted according to the VIEW, ZONE, and LEFT/RIGHT specifications. In other words, if data within a line is modified, only that data is modified in the file unless the data modified is outside the current zone. On the other hand, if the line is found to contain logical tab characters and tabbing is in effect (tab positions are set), the input is interpreted as a complete replacement line in a column 1 through 80 format regardless of the VIEW or LEFT/RIGHT specifications. This, of course, facilitates the entry of new lines to the file via the LADD editor command or the 'A' editor line command regardless of VIEW or LEFT/RIGHT settings.

When the REPOPTION option is set ON, the checking for tab characters is bypassed and all data is treated alike as modifications to the records on the screen. This does not mean, however, that the tab characters themselves will not be interpreted.

Note: The following editor command versions of the [|]SET system command (see "[|]SET Command" on page 3-111) will be frequently used in the full screen editor environment.

```
SET PFnnED  (Set the meaning of a PF key in the edit set)
SET END=    (Set the logical line end character)
SET ESC=    (Set the escape character)
SET TAB=    (Set the logical tab character)
SET BYPASS  (Set control character interpretation ON or OFF)
SET DATANL  (Set APL or Text keyboard feature ON or OFF)
SET LOG     (Set logging ON or OFF)
```

SHIFT Command

The SHIFT command shifts the data within the current zone to the left or right a given number of columns.

```
SHIft        [Left|Right [nn|1]]
```

nn is a decimal number indicating the span of the shift operation.

The column suffix can be used with this command.

The direction of the shift is indicated by the LEFT or RIGHT operands. If the first operand is omitted, LEFT is assumed. The number of columns or span of the shift is indicated by the 'nn' operand. If this operand is omitted, 1 is assumed.

Only the data within the zone defined by the ZONE command or the column suffix, if present, participates in the shift operation. Any data shifted out of the zone is lost. Space characters (blanks) are inserted to replace data shifted out of the zone. If the shift amount is greater than or equal to the size of the zone, the entire zone will end up as spaces.

If the data within several lines is to be shifted, the REPEAT command can be entered prior to the SHIFT command to indicate the number of lines to participate in the operation.

Example:

Shift three lines left by 10 positions. Notice how the REPEAT command is used to determine the number of executions of the SHIFT command.

```
===> next;repeat 3;shift left 10;top_
<<..+....1....+....2....+....3. .5....+.. MEM=SHIFTTXT>>..+..FS
This meaningless text example contains a few lines       /===/*
                   which were indented much too          *===*
                   far such that they disturb the        *===*
                   nice appearance of the document.      *===*
We shall move them left using the 'SHIFT' command.       *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=SHIFTTXT>>..+..FS
***** TOP OF FILE *****                                   /***/
This meaningless text example contains a few lines        *===*
         which were indented much too                     *===*
         far such that they disturb the                   *===*
         nice appearance of the document.                 *===*
We shall move them left using the 'SHIFT' command.        *===*
```

SHOW Command

The SHOW command displays all settings that can be set via the /SET system command. In addition, under the full screen editor only, the SHOW command allows you to display the names of the files being edited concurrently.

```
   SHow          NAMes
```

The display contains the name of each file being edited, that is, the names of those files which were specified via the /EDIT or ENTER commands and which have not as yet been quit or filed. The display also indicates whether the file is a member of the library ('LIB'), a newly created file ('NEW') or is the input area ('INP'). In addition, the display indicates whether or not the file is presently associated with a logical screen.

After the response to the SHOW command, press ENTER to restore the full screen editor formatted screen.

Example:

```
===> show names_
<<..+....1....+....2....+....3. .5....+.. MEM=MEMBER2 >>..+..FS
***** TOP OF FILE *****                                   /***/
RECORD NUMBER  1 OF MEMBER2                               *===*
RECORD NUMBER  2 OF MEMBER2                               *===*
RECORD NUMBER  3 OF MEMBER2                               *===*
```

```
 _
 ...+....1....+....2....+....3. .5....+....6....+....7....+..ED
*
NAME      TYPE  SCREEN
*
MEMBER1    LIB
MEMBER2    LIB    YES
NEWPROG    NEW
```

SPLIT Command

The SPLIT command is used during text editing to split the current line into two lines.

```
SPlit        /string/|nn
```

/string/	is the character string in front of which the line split is to occur.
nn	is the column number in front of which the line split is to occur. Its range is 1 to 80.

The column suffix can be used with this command.

The location at which the split is to occur can be specified by a character string or a column number.

If /string/ is specified, the line is split before the string; the first occurrence of the string in the line becomes the start of the second line.

Similarly, if 'nn' is specified, the data at column 'nn' becomes the start of the second line.

All data up to the split point is retained as the first line; all data from the split point on becomes the second line. The data in the second line is aligned in the current zone.

Example:

Split the current line at the first occurrence, in columns 30 and above, of the string 'The'.

```
===> next;splitc30 /The/;top_
<<..+....1....+....2....+....3. .5....+.. MEM=SPLITTXT>>..+..FS
***** TOP OF FILE *****                                   /***/
The text shown here is incomplete. The second             *===*
sentence needs some introductory information              *===*
to be understandable. . . . . . . . . . . .               *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=SPLITTXT>>..+..FS
***** TOP OF FILE *****                                   /***/
The text shown here is incomplete.                        *===*
The second                                                *===*
sentence needs some introductory information              *===*
to be understandable. . . . . . . . . . . .               *===*
```

```
===>
<<..+....1....+....2....+....3. .5....+.. MEM=SPLITTXT>>..+..FS
***** TOP OF FILE *****                                   /***/
The text shown here is incomplete. HERE IS THE            *===*
IMPORTANT PREREQUISITE INFORMATION._The second            *===*
sentence needs some introductory information              *===*
to be understandable. . . . . . . . . . . .               *===*
```

STACK Command

The STACK command places commands or data into the editor stack. The command can also be used to control the placement of data within the stack and to preserve editor settings.

```
STACk          OPEN
               CLOSE
               BACK [mm]
               EDIT
               /string/
               nn|0
               data
```

OPEN	Allocates and logically opens the stack, if it does not already exist. Sets the stack line pointer to the top, so that the next line of input to the stack will be line 1.
CLOSE	Logically closes the stack by marking the end with a terminator. If more commands are written to the stack, the terminator is overwritten.
BACK [mm]	Moves the stack line pointer back 'mm' lines (the default is 1). This operand can be used to delete entries from the stack. For example, if you write an incorrect line to the stack, STACK BACK 1 would cause the following lines to overlay the incorrect one.
EDIT	Writes a line containing editor settings, control characters and tab positions to the stack. Command settings that can be stored are:

```
CASE     IMAGE        VERIFY
DELIM    LINEMODE     ZONE
FLAG     TABSET       SET (control characters only)
```

The settings can be restored with the RESTORE command.

/string/	A character string indicating that all lines down to (but not including the line containing /string/) are to be placed in the stack. The maximum is 100. Unlike the other STACK options, /string/ does not extend the stack if a 'stack full' condition occurs. Thus, an overrun condition does not occur if /string/ is entered incorrectly. /string/ can contain double-byte characters.
nn|0	Causes 'nn' lines (1 to 99) to be placed in the stack, beginning with the current line. You can specify '0' (null line) to terminate input mode when you are stacking commands for later execution.
data	Lets you write data and/or commands to the stack. These commands can be executed by specifying the stack as if it were a macro command. For example: @$$STACK data can contain double-byte characters.

The stack is a work area for use during editing. Typically, it is used as a temporary save area when you are moving or copying data (M and C commands) within a file. The stack is also a temporary holding area for editor macros such as @MOVE and @COPY.

The stack is identical with the punch area that is used during executions. Thus, it will still contain the output from the previous interactive partition execution when you enter edit mode. Lines stacked during editing remain in the stack after edit mode is terminated.

Because the stack and the punch area are the same, do not use the stack while editing in asynchronous execution mode (/ASYNCH command). You could come into conflict with the job running in the interactive partition.

Example:

Store several statements in the stack which are later to be included in an assembler source program.

You enter the following sequence of STACK commands:

```
stack open
stack ins           eject
stack ins *------------------------------------*
stack ins *           &&PARAM1 routine          *
stack ins *------------------------------------*
stack ins           space 3
stack close
```

Issuing the EDPUN macro gives the following display of the stack area:

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=$$PUNCH >>..+..FS
***** TOP OF FILE *****                                    /***/
INS           EJECT                                        *===*
INS *--------------------------------------*               *===*
INS *           &&PARAM1 ROUTINE           *               *===*
INS *--------------------------------------*               *===*
INS           SPACE 3                                      *===*
```

Now edit member SAMPASMB and, with the @$$STACK macro, retrieve the statements from the stack. Notice how the '/' line command is used to select the 5th line as the insert point. The parameter 'readcard' in the @$$STACK macro call will be the value of &&PARAM1.

```
===> @$$stack readcard;up 5_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
MAINPGM  CSECT                                             *===*
         PRINT GEN                                         *===*
         BALR  5,0                                         *===*
         USING *,5                                         *===*
         OPEN  FILOUT                                      /===*
LOOPA    EXCP  RDCCB                                       *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
         OPEN  FILOUT                                      /===/*
          EJECT                                            *===*
*--------------------------------------*                   *===*
*          READCARD ROUTINE            *                   *===*
*--------------------------------------*                   *===*
          SPACE 3                                          *===*
LOOPA    EXCP  RDCCB                                       *===*
```

STATUS Command (see [/]SHOW System Command)
TABSET Command(see [/]TABSET System Command)

TOP Command

The TOP command positions the pointer to the top of the file (to the null line in front of the first line in the file).

```
Top
```

An automatic TOP is performed by the FIND, LOCATE, and CHANGE requests if an end-of-file condition immediately preceded the request. An automatic TOP is also performed by the SEARCH command.

The INPUT or INSERT commands can be used following the TOP command to insert new lines before the first line in the file.

Example:

```
    request:          t
    effect:           (Pointer is positioned to the top of the file.)
```

TYPE Command (see PRINT Command)

UP Command

The UP command repositions the pointer 'n' lines before the current line.

```
Up              [n|1]
```

n indicates the number of lines by which the pointer is to be moved back. The default is 1 and the maximum is 99999. If top-of-file is reached before the line pointer has moved up n lines, the pointer is positioned at the first line and the INVALID RANGE message appears.

If n is not specified, a value of 1 is assumed, and the pointer is moved up to the previous line in the file.

After the command has been executed successfully, the new current line is displayed. To move the pointer to the top of the file, issue the TOP command.

Example:

```
u 9
```

This request repositions the pointer nine lines before the current line.

VERIFY Command

The VERIFY command is primarily used to vary the settings of two options: (1) the number of lines displayed prior to the current line, and (2) the number of physical lines associated with the command area.

```
Verify        [FULL] [n1 [n2]]
```

FULL	This operand is optional. Its only purpose is to retain format compatibility with the VERIFY command issued from the context editor.
n1	is a decimal number greater than zero which indicates the number of lines to be displayed prior to and including the current line.
n2	is a decimal number (1, 2, 3, or 4) indicating the number of physical lines to be associated with the command area for the file being edited.

The VERIFY command displays the complete contents of the physical screen regardless of whether or not any operands are specified.

The 'n1' operand specifies the number of logical records to be displayed (within a format area) prior to and including the current line. The number specified can be from 1 to 16. Specifying '1' would indicate that no records prior to the current line are to be displayed. Assuming that the format area contained 21 records, this would mean that the current line and the next 20 lines would be displayed. On the other hand, specifying 'n1' as '9' would indicate that 8 records prior to and 12 records following the current line would be displayed.

The 'n2' operand specifies the number of physical screen lines to be associated with the command area. 1, 2, 3, or 4 can be specified.

Note: *It is possible to specify a value of 'n1' such that the current line does not appear on the screen. For example, if a format area contained space for 10 records and the 'n1' value was specified as 12, the current line would actually be 2 lines beyond the end of the format area, thus, not on the screen.*

VIEW Command

The VIEW command is used to rearrange, format, and view the 80 character record on the screen.

```
VIEW        [[H]m n][,[H]m n][,[H]m n] ...
```

m   is an integer from 1 to 80 which specifies the starting column of the field to be displayed. (If 'm' is zero, a filler field of spaces is assumed.)

n   is an integer from 1 to 80 which specifies the ending column number of the field to be displayed. (If 'm' is zero, 'n' is the number of filler spaces to be inserted.)

The VIEW command lets you:

- View only certain columns of the 80-character record on the screen. For example, to only view columns 12 through 80.
- Rearrange the order in which data is displayed. For example, to view columns 41 through 80 followed by columns 1 through 40.
- View certain fields in hexadecimal format rather than in character format.
- Place blank filler fields between fields on the screen.

If the VIEW command is entered with no operands, the current VIEW settings are displayed in the command area associated with the logical screen.

The normal VIEW specification at the time a file is entered is 'VIEW 1 80' which says view columns 1 through 80 in character format. Thus, the default view contains one 80-character field.

To vary the default view, specify pairs of operands 'm' and 'n' representing the beginning and ending column numbers of the fields to be displayed. For example, if you want to only view and modify columns 41 through 80 of each record, you would specify 'VIEW 41 80'. Or, to see columns 41 through 80 at the left-hand side of the screen and columns 1 through 40 at the right-hand side of the screen, you would specify 'VIEW 41 80,1 40'.

Up to 12 fields can be specified in this way. Fields can overlap, in which case if you want to modify an overlapping field, you must make the modification to the rightmost occurrence of the field for the modification to be effective.

The VIEW specifications apply to the file being edited when the VIEW command is entered. Thus, different files being edited at the same time can all have different VIEW specifications.

If the letter 'H' immediately precedes an 'm' operand, the field will be displayed on the screen in hexadecimal format. Any modifications to the field as it is displayed on the screen must also be made in hexadecimal format. Note that a hexadecimal format field will require twice as many columns on the screen as there are columns within the actual field. If the total length of the viewed fields exceeds the length of one output line, use the FORMAT command to increase the number of output lines per displayed record.

If the 'm' operand is specified as zero, a filler field is assumed and 'n' blanks will be inserted in the display line. For example, the command 'VIEW 1 10,0 5,11 20,0 5,21 40' will cause three fields to be displayed separated from one another by 5 blank columns.

The VIEW command displays mixed data correctly, as long as

- Only the first field requests the display of characters.
- All fields following the first field are hexadecimal or filler fields.

If a subfield of double-byte characters exceeds the size of the specified area, the leftmost or rightmost double-byte character will be replaced by one or two control characters, which guarantee that the remainder of the subfield is displayed correctly. The data in the VSE/ICCF library is not affected by this modification, as long as you do not

- Replace the added control characters
- Change double-byte characters to one-byte characters.

Examples:

Example 1:

View columns 10 through 80 thus making visible the information that is hidden behind the line command area.

```
===> view 10 80_
<<..+....1....+....2....+....3. .5....+.. MEM=INVENTRY ...+..F>
***** TOP OF FILE *****                                   /***/
ITEM1 DESCRIPTION       5   PUMPS & PIPES LTD., HUSTON    *===*
ITEM2 DESCRIPTION      28   JAMES & JONES, NEW YORK       *===*
ITEM3 DESCRIPTION       2   MODERN PLASTICS, DALLAS       *===*
ITEM4 DESCRIPTION    2378   CANADIAN PLYWOOD FACTORY, TO *===*
ITEM5 DESCRIPTION   28355   CHEMICAL INSTRUMENTS, ALABAM *===*
ITEM6 DESCRIPTION     289   MASSATRONICS LTD., MASSACHUS *===*
```

```
===> _
<<..+....1....+....2....+....3. .5....+.. MEM=INVENTRY ...+..F>
***** TOP OF FILE *****                                   /***/
CRIPTION        5   PUMPS & PIPES LTD., HUSTON            *===*
CRIPTION       28   JAMES & JONES, NEW YORK               *===*
CRIPTION        2   MODERN PLASTICS, DALLAS               *===*
CRIPTION     2378   CANADIAN PLYWOOD FACTORY, TORONTO     *===*
CRIPTION    28355   CHEMICAL INSTRUMENTS, ALABAMA         *===*
CRIPTION      289   MASSATRONICS LTD., MASSACHUSETTS      *===*
```

Example 2:

Display part of an object module in EBCDIC and hexadecimal format. First the screen is formatted to reserve three screen lines for every line in the file. Then the VIEW command is issued to partition the three lines as follows:

1. Columns 1-72 and 74-80 in EBCDIC in the first line (VIEW 1 72,74 80). Column 73 is sacrificed in order to make the sequence number fully visible.
2. In the second line the storage address (VIEW H6 8) plus the contents in columns 17-46 (VIEW H17 46).
3. In the third line the ESDID (VIEW H15 16) plus the contents in columns 47-72 (H47 72).

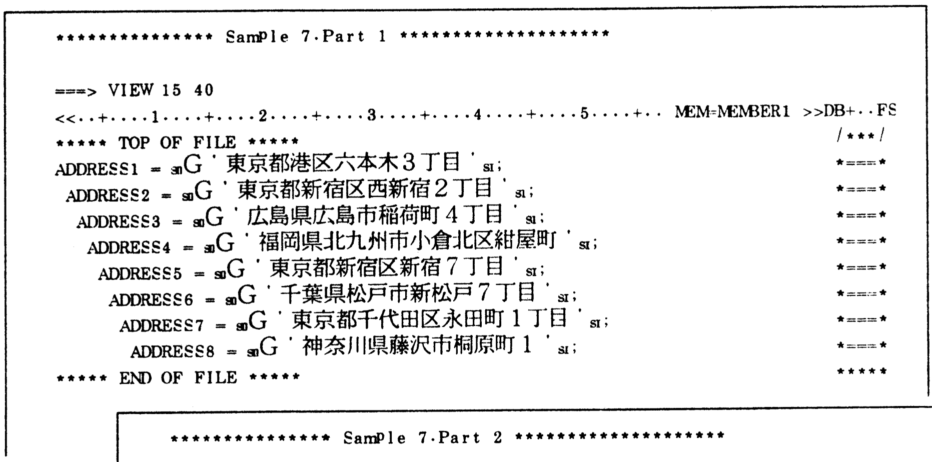
```
===> format 3_
<<..+....1....+....2....+....3. .5....+.. MEM=TPROGOBJ>>..+..FS
 ESD              TPROG             IJ2M0011                /===/*
 TXT                  K   Ys K     h&          &  D       s   1      *===*
 TXT                        Dq        $ s      u       s g    n      *===*
 TXT    Y                            s                               *===*
```

```
===> VIEW 1 72,74 80,H6 8,0 4,H17 46,0 12,H15 16,0 4,H47 72
<<..+....1....+....2....+....3. .5....+.. MEM=TPROGOBJ>>..+..FS
***** TOP OF FILE *****

                                                              /***/
  ESD              TPROG             IJ2M0011                   000
1

                                                              *===*
  TXT                  K   Ys K     h&         &  D      s  1   000
2

                                                              *===*
  TXT                       Dq        $ s      u      s g   n   000
3

                                                              *===*
  TXT    Y                       s  1                           000
4
```

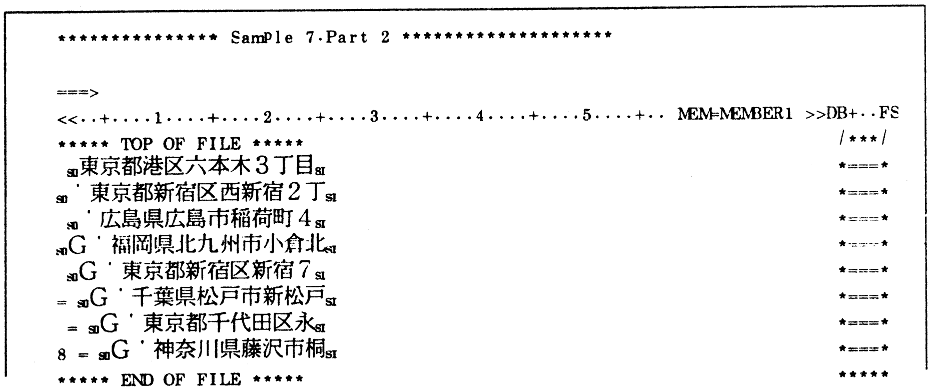
```
 ===> _
 <<..+....1....+....2....+....3. .5....+.. MEM=TPROGOBJ>>..+..FS
  ESD          TPROG            IJ2M0011                   0001
404040    E3D7D9D6C7404040000000 3F0F0F1F0000000004000
  0001    0068C9D1F2D4F0F0F1F100 0404040404040            /===/*
  TXT             K   Ys K     h&        &  D      s  1    0002
000078    05A0D217A0E8A22ED237A1 0001858F0A24605EF5820
  0001    A1DC0A0E90EAD00C18AF50 0A22A58F10010            *===*
  TXT                  Dq        $ s     u      s g   n    0003
0000B0    45EF000C58D0A1C498EAD0 F50D0A1A041D0A19C5800
  0001    A1945B00A20A4120A0A418 0A02E95402000            *===*
  TXT   Y                      s  1                        0004
0000E8    4780A0221B214122000158 F000C58D0A1A098EAD00C
  0001    07FE000080000000000400 0000003000000            *===*
```

Example 3:

Display mixed data in columns 15 through 40.



```
*************** SamPle 7·Part 1 ********************

===> VIEW 15 40
<<··+····1····+····2····+····3····+····4····+····5····+·· MEM=MEMBER1 >>DB+··FS
***** TOP OF FILE *****                                                  /***/
ADDRESS1 = SOG ' 東京都港区六本木３丁目 ' SI;                              *===*
 ADDRESS2 = SOG ' 東京都新宿区西新宿２丁目 ' SI;                           *===*
  ADDRESS3 = SOG ' 広島県広島市稲荷町４丁目 ' SI;                          *===*
   ADDRESS4 = SOG ' 福岡県北九州市小倉北区紺屋町 ' SI;                     *===*
    ADDRESS5 = SOG ' 東京都新宿区新宿７丁目 ' SI;                          *===*
     ADDRESS6 = SOG ' 千葉県松戸市新松戸７丁目 ' SI;                       *===*
      ADDRESS7 = SOG ' 東京都千代田区永田町１丁目 ' SI;                    *===*
       ADDRESS8 = SOG ' 神奈川県藤沢市桐原町１ ' SI;                       *===*
***** END OF FILE *****                                                  *****
```



```
*************** SamPle 7·Part 2 ********************

===>
<<··+····1····+····2····+····3····+····4····+····5····+·· MEM=MEMBER1 >>DB+··FS
***** TOP OF FILE *****                                                  /***/
 SO東京都港区六本木３丁目SI                                               *===*
SO ' 東京都新宿区西新宿２丁SI                                             *===*
 SO ' 広島県広島市稲荷町４SI                                              *===*
SOG ' 福岡県北九州市小倉北SI                                              *===*
 SOG ' 東京都新宿区新宿７SI                                               *===*
= SOG ' 千葉県松戸市新松戸SI                                              *===*
 = SOG ' 東京都千代田区永SI                                               *===*
8 = SOG ' 神奈川県藤沢市桐SI                                              *===*
***** END OF FILE *****                                                  *****
```

ZONE Command

The ZONE command causes subsequent locating commands (such as FIND, LOCATE, SEARCH) and alteration commands (such as CHANGE, ALTER, REWRITE, OVERALAYX) to apply only to the specified zone of the lines.

```
Zone           [n1|*|1 [n2|*|72|80]]
```

n1 specifies the initial column of the zone of each line which is to be scanned. The default value is column 1. If * is specified, column 1 is assumed.

n2 specifies the final column of the zone of each line which is to be scanned. If n2 is omitted or specified as *, column 72 or 80 (depending on an installation option) is assumed. If n2 is specified, n1 must also be specified.

The initial zone column, n1, must be less than or equal to the final zone column, n2, and n2 must be less than or equal to 80.

When you are editing mixed data, the modification of data on the screen after a ZONE command can lead to unwanted results, as in the following cases:

- If before or after the modification the zone column cuts across a double-byte character. This is illustrated by Example 2, 3, and 4.
- If alphanumeric data is replaced by a string of double-byte characters. In this case the alphanumeric data outside the zone is retained in their original form, but the replacement string of double-byte characters is restructured to fit inside the zone. This case is illustrated by Example 5.
- If double-byte characters are replaced by a string of alphanumeric characters. In this case, only those double-byte characters are retained that can be fit inside a pair of SO and SI characters. Example 6 illustrates this case.

The zone values remain in effect until the next ZONE command is issued, or until the edit session is terminated. These values can also be temporarily overridden with the column suffix Cnn (see "Limiting Editor Operations to Certain Columns" on page 4-14).

When editing multiple files concurrently, the zone can be set for each file being edited.

In addition to its obvious uses in searching and modifying fixed-format 80-character line files, the ZONE command is useful in source program editing. For example, it can be used to add comment fields, continuation characters, and for searching the serialization columns 73-80. Note that, if editor change flagging is on, only the VSE/ICCF administrator can extend a search into the flag field. Searching in the editor flag field is not allowed for any other user.

Examples:

Example 1:

Change the string 'LOOP' to 'LOOPA' in all lines. Restrict the change to columns 1 to 30 (ZONE 1 30).

```
===> zone 1 30;change /loop /loopa/ *;search /loopa/_
<<..+....1....+....2....+....3. .5....+.. MEM=SAMPASMB>>..+..FS
          SPACE 3                                          /===/*
LOOP     EXCP  RDCCB         READ CARD                     *===*
         WAIT  RDCCB                                       *===*
         CLC   R(3),=C'/* '  LAST CARD?                    *===*
         BE    ENDCARD       EXIT LOOP IF YES              *===*
         MVC   0(80,2),R     CARD— TO DISK BUFFER          *===*
         PUT   FILOUT        WRITE TO DISK                 *===*
         B     LOOP          LOOP UNTIL "/*" CARD          *===*
ENDCARD  CLOSE FILOUT                                      *===*
```

```
===> _
<<..+....1....+....2....+...>>. .5....+.. MEM=SAMPASMB ...+..FS
LOOPA    EXCP  RDCCB         READ CARD                     /===/*
         WAIT  RDCCB                                       *===*
         CLC   R(3),=C'/* '  LAST CARD?                    *===*
         BE    ENDCARD       EXIT LOOP IF YES              *===*
         MVC   0(80,2),R     CARD— TO DISK BUFFER          *===*
         PUT   FILOUT        WRITE TO DISK                 *===*
         B     LOOPA         LOOP UNTIL "/*" CARD          *===*
ENDCARD  CLOSE FILOUT                                      *===*
```

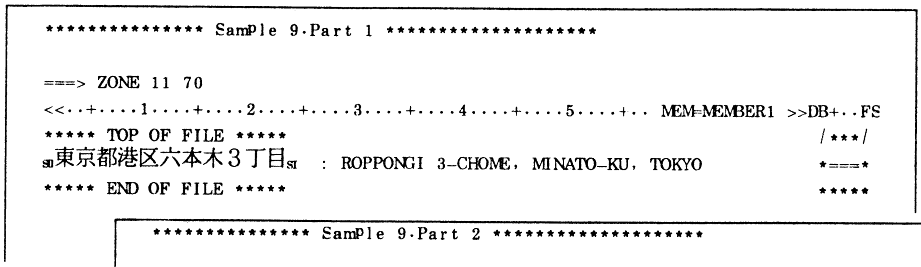
Example 2:

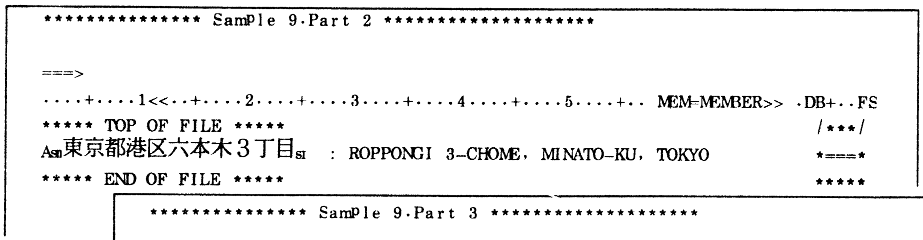
```
*************** Sample 8-Part 1 ********************

===> ZONE 15 72
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                              /***/
東京都港区六本木３丁目 : ROPPONGI 3-CHOME, MINATO-KU, TOKYO           *===*
***** END OF FILE *****                                              *****
```

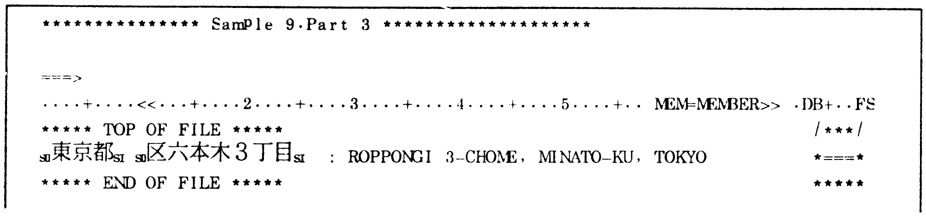
```
*************** Sample 8-Part 2 ********************

===>
....+....1....<<...2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                              /***/
東京都港区南麻布３丁目 : MINAMI-AZABU 3-CHOME, MINATO-KU, TOKYO       *===*
***** END OF FILE *****                                              *****
```

```
*************** Sample 8-Part 3 ********************

===>
....+....1....<<...2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                              /***/
東京都港区六海布３丁目 : MINAMI-AZABU 3-CHOME, MINATO-KU, TOKYO       *===*
***** END OF FILE *****                                              *****
```

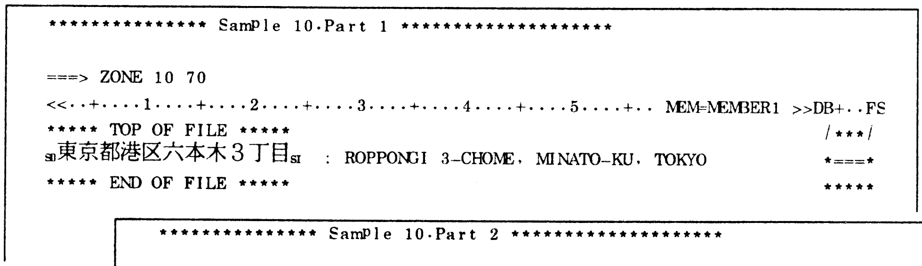
Example 3:

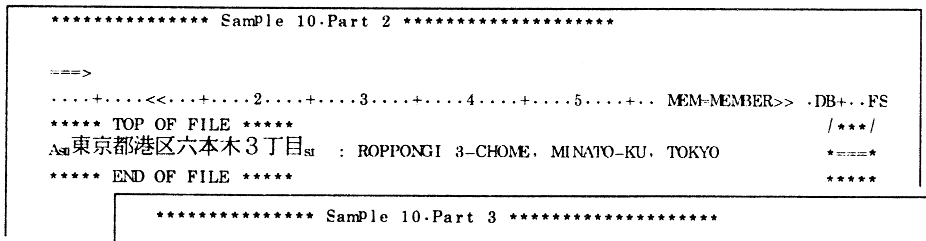

```
*************** SamPle 9·Part 1 ********************

===> ZONE 11 70
<<··+····1····+····2····+····3····+····4····+····5····+·· MEM=MEMBER1 >>DB+··FS
***** TOP OF FILE *****                                                  /***/
SO東京都港区六本木３丁目SI : ROPPONGI 3-CHOME, MINATO-KU, TOKYO               *===*
***** END OF FILE *****                                                  *****
```

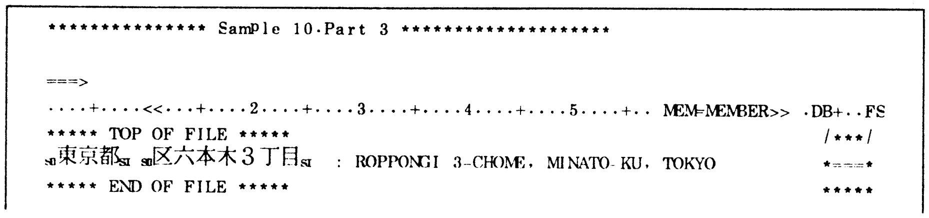

```
*************** SamPle 9·Part 2 ********************

===>
····+····1<<··+····2····+····3····+····4····+····5····+·· MEM=MEMBER>> ·DB+··FS
***** TOP OF FILE *****                                                  /***/
ASO東京都港区六本木３丁目SI : ROPPONGI 3-CHOME, MINATO-KU, TOKYO              *===*
***** END OF FILE *****                                                  *****
```


```
*************** SamPle 9·Part 3 ********************

===>
····+····<<···+····2····+····3····+····4····+····5····+·· MEM=MEMBER>> ·DB+··FS
***** TOP OF FILE *****                                                  /***/
SO東京都SI SO区六本木３丁目SI : ROPPONGI 3-CHOME, MINATO-KU, TOKYO            *===*
***** END OF FILE *****                                                  *****
```

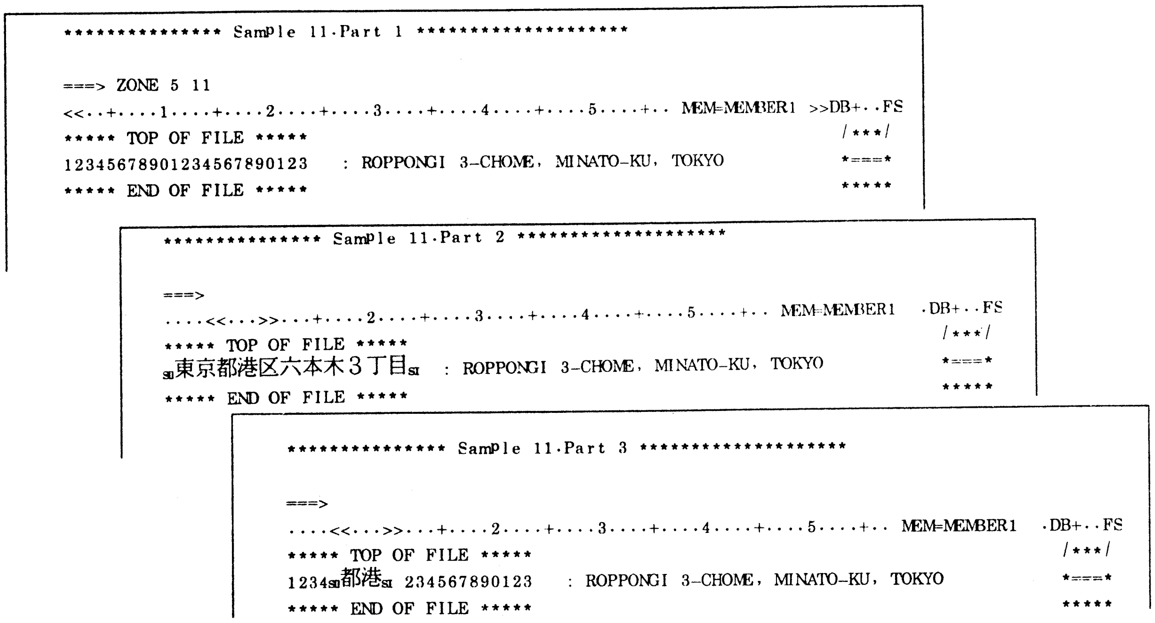
Example 4:


```
*************** SamPle 10·Part 1 ********************

===> ZONE 10 70
<<··+····1····+····2····+····3····+····4····+····5····+·· MEM=MEMBER1 >>DB+··FS
***** TOP OF FILE *****                                                  /***/
SO東京都港区六本木３丁目SI : ROPPONGI 3-CHOME, MINATO-KU, TOKYO               *===*
***** END OF FILE *****                                                  *****
```


```
*************** SamPle 10·Part 2 ********************

===>
····+····<<···+····2····+····3····+····4····+····5····+·· MEM=MEMBER>> ·DB+··FS
***** TOP OF FILE *****                                                  /***/
ASO東京都港区六本木３丁目SI : ROPPONGI 3-CHOME, MINATO-KU, TOKYO              *===*
***** END OF FILE *****                                                  *****
```


```
*************** SamPle 10·Part 3 ********************

===>
····+····<<···+····2····+····3····+····4····+····5····+·· MEM=MEMBER>> ·DB+··FS
***** TOP OF FILE *****                                                  /***/
SO東京都SI SO区六本木３丁目SI : ROPPONGI 3-CHOME, MINATO-KU, TOKYO            *===*
***** END OF FILE *****                                                  *****
```

Example 5:

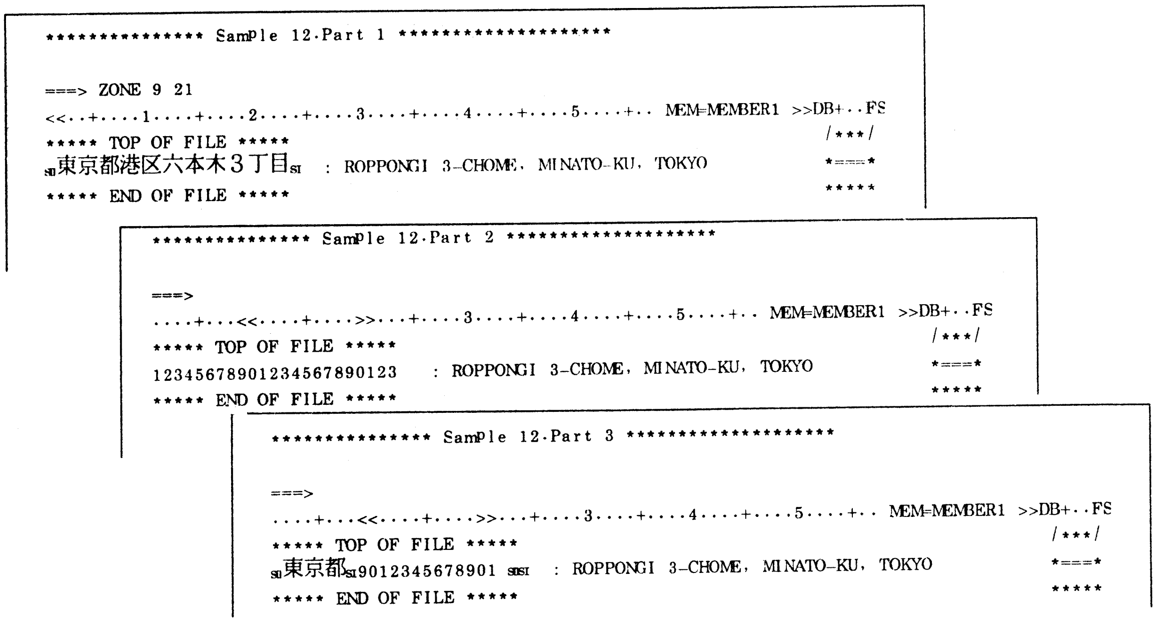


```
*************** Sample 11-Part 1 *******************

===> ZONE 5 11
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                                /***/
12345678901234567890123   : ROPPONGI 3-CHOME, MINATO-KU, TOKYO         *===*
***** END OF FILE *****                                                *****
```

```
*************** Sample 11-Part 2 *******************

===>
....<<...>>...+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 .DB+..FS
***** TOP OF FILE *****                                                /***/
SO東京都港区六本木３丁目SI   : ROPPONGI 3-CHOME, MINATO-KU, TOKYO       *===*
***** END OF FILE *****                                                *****
```

```
*************** Sample 11-Part 3 *******************

===>
....<<...>>...+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 .DB+..FS
***** TOP OF FILE *****                                                /***/
1234SO都港SI 234567890123   : ROPPONGI 3-CHOME, MINATO-KU, TOKYO       *===*
***** END OF FILE *****                                                *****
```

Example 6:



```
*************** Sample 12-Part 1 *******************

===> ZONE 9 21
<<..+....1....+....2....+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                                /***/
SO東京都港区六本木３丁目SI   : ROPPONGI 3-CHOME, MINATO-KU, TOKYO       *===*
***** END OF FILE *****                                                *****
```

```
*************** Sample 12-Part 2 *******************

===>
....+...<<....+....>>...+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                                /***/
12345678901234567890123   : ROPPONGI 3-CHOME, MINATO-KU, TOKYO         *===*
***** END OF FILE *****                                                *****
```

```
*************** Sample 12-Part 3 *******************

===>
....+...<<....+....>>...+....3....+....4....+....5....+.. MEM=MEMBER1 >>DB+..FS
***** TOP OF FILE *****                                                /***/
SO東京都SI9012345678901 SOSI   : ROPPONGI 3-CHOME, MINATO-KU, TOKYO    *===*
***** END OF FILE *****                                                *****
```

'nn' Command

The 'nn' command adds, replaces or locates lines by sequence number when linenumber editing is in effect.

```
'nn'          [line]
```

nn	is a decimal number of up to eight digits referencing a sequence number within the file being edited. Leading zeros can be omitted.
line	specifies a line of data to be inserted into the file at the specified sequence number. If the sequence number already exists within the file, the existing line is replaced with the new data 'line'. If 'line' is omitted, the line pointer is positioned to the sequence number specified.

After the command has been executed the line pointer is positioned to the inserted, replaced or located line.

A sequence number (as the command) followed by a line of data (line operand) causes the corresponding line in the file to be replaced with the new data. If the sequence number does not exist within the file, the new line is inserted at the appropriate location within the file. Thus, this command can be used to make replacements or additions to a file when linenumber editing is in effect without having to position the line pointer.

If a sequence number is specified but the line operand is omitted, the line pointer will be positioned to the line containing the specified sequence number. If this sequence number does not exist within the file, the line pointer will not be moved and a 'LINE NOT FOUND' message will be displayed. (The 'LINE NOT FOUND' message can also occur if the program that you are editing is out of sequence.)

If indexed editing is in effect and linenumber editing had been set prior to issuing the INDEX command (building the index), sequence numbers will be retained in the index. Thus, using the 'nn' command to set the line pointer provides a concise and very fast method of moving the line pointer long distances.

Even though linenumber editing is in effect, you can still use the INPUT, INSERT and REPLACE commands where these are more convenient. They are usually more convenient when the line pointer has already been positioned to the desired location. On the other hand, the 'nn' command will both position the line pointer and insert or replace the specified line.

Example:

Replace the contents of line 60, add a line between lines 70 and 80, make line 20 the current line.

```
===> linemode left;60 print i,j,k;75 print ' ';20_
....+<<..1....+....2....+....3. .5....+.. MEM=CALC     ...+..F>
***** TOP OF FILE *****                                  /***/
00010PRINT 'ENTER THREE NUMBERS SEPARATED WITH COMMAS'   *===*
00020PRINT 'ENTER /* TO TERMINATE'                       *===*
00030L = I + J + K                                       *===*
00040M = I * &SQR2                                       *===*
00050N = J * &PI                                         *===*
00060PRINT I,K,K                                         *===*
00070PRINT L,M,N                                         *===*
00080PRINT 'END OF CALCULATION'                          *===*
```

```
===> _
....+<<..1....+....2....+....3. .5....+.. MEM=CALC     ...+..F>
00020PRINT 'ENTER /* TO TERMINATE'                      /===/*
00030L = I + J + K                                       *===*
00040M = I * &SQR2                                       *===*
00050N = J * &PI                                         *===*
00060PRINT I,J,K                                         *===*
00070PRINT L,M,N                                         *===*
00075PRINT ' '                                           *===*
00080PRINT 'END OF CALCULATION'                          *===*
00090GO TO 10                                            *===*
00100END                                                 *===*
***** END OF FILE *****                                  *****
```

Editor Line Commands

Line commands carry out editing functions for individual or multiple lines on the screen. They do not necessarily reference or alter the line pointer. The following commands apply to both alpahnumeric and mixed data:

- ADD
- COPY
- DELETE
- DUPLICATE
- INSERT
- MOVE
- SET LINE POINTER
- STACK

All other editor line commands only process alpanumeric data.

Line Command Area

Individual lines within a format area on the screen will have line command areas associated with them unless these command areas have been deleted by the FORMAT command. Line commands are entered into these line command areas. A command entered into the line command area applies to the record associated with the line command area and optionally to a number of records following the record for which the line command was entered.

The initial contents of the line command area will be one of the following:

1. The characters '*= = =*' or '/= = =/*' will appear in the line command area for displayed records if the NUMBERS option (see the SET editor command on page 4-122) is set off. Because OFF is the default setting, one of these two identifiers will always appear unless SET NUMBERS ON has been issued. The current line displays '/= = =/*', all others '*= = =*'.

2. The characters 'nnnnnn' will appear in the line command area where 'nnnnnn' is the sequence number found within columns 73 through 80 or in the LINEMODE columns if the SET NUMBERS ON command has been specified for the file being edited.

3. The characters '*INPUT' will appear in the line command area after an INPUT or INSERT command has been entered for lines which have been formatted to receive new input.

4. The characters '/***/' will appear in the line command area of the line associated with the 'top of file' and '*****' with the 'bottom of file' indicator. The line command area for the bottom of the file is non-functional; however, the line command area for the top of the file can be used to enter the 'A' (add) or 'I' (insert) commands for adding new lines prior to the first record in the file.

Entering Line Commands

Line commands consist of one or two characters and may or may not have a single numeric operand (two line commands, '>' and '<', may actually have two numeric operands). The numeric operand(s) can precede or follow the command code. Numeric operands can range from 1 to 999 unless the NUMBERS option is set, in which case the range is 1 to 99. If a numeric operand is omitted, a value of '1' is assumed. Only one command per line command area is permitted, except in the case of the '/' command which can precede another command.

If the characters in the line command area are '*= = =*' or '/= = =/*' or '*****', the line command can be entered beginning at any position within the line command area. Blanks or commas can be used as delimiters within the area although no delimiters are required except in the case of the '>' and '<' commands with two operands, where the two numeric operands must be separated by a delimiter (blank, comma, equal sign or asterisk).

If the characters in the command area are actual columns from the file because the NUMBERS option is set on, some special rules apply. First, the line command must begin in position 1 of the line command area. Secondly, only one numeric operand is allowed and it must immediately precede or follow (no intervening delimiters) the line command code itself. Thirdly, a blank must be entered following the command (and operand, if there is one) unless the operand precedes the command itself. This is a safety precaution which helps avoid the numbers in this area being treated as an operand, for example as a 'D' (delete) command which can cause a damaged file.

Line commands can be entered on several lines prior to pressing ENTER. If an error occurs in a line command area, a message is displayed and the audible alarm sounds. If multiple commands were entered, the only indication of which command(s) failed would be whether or not the requested function had actually been carried out.

`A[nn]` ADD

The 'A' (add) command can be used to add (from 1 to 999) blank lines after the line on which the command is issued. If no operand is specified, '1' is assumed. The blank lines added to the file can be used for the entry of new data to the file. When ENTER is pressed, the specified number of blanks will be added to the file and the cursor will be positioned to column 1 of the first added line to facilitate the addition of new data (see the LADD command for information concerning the actual typing in of the new data).

`C[nn]` COPY

The 'C' (copy) command places 'nn' (1-99) lines, beginning with the line on which it is entered, into the stack area for insertion elsewhere in the file, or in a different file. The point at which the data is to be inserted can be indicated on the same screen or on some later screen by the 'I' (insert) line command. (Functionally, a 'C' or 'M' command causes a STACK OPEN followed by the placing of the lines in the stack. Compare this with the 'K' command below which does not cause the stack to be opened).

Note: *The C (copy), I (insert), K (copy to stack), and M (move) commands use the stack area, which is the same as the punch area. In asynchronous processing mode there is thus a risk that the data handled by these commands could be destroyed by the running program when it uses the punch (=stack) area for punching.*

`D[nn]` DELETE

The 'D' (delete) command can be used to delete (from 1 to 999) lines from the file, beginning with the line on which it is issued. If no operand is specified, '1' is assumed. If multiple format areas are defined within a logical screen, deletions can only be made to one of the formats prior to pressing the ENTER key. If the current line is deleted, the line pointer is advanced to the next line within the file.

I	INSERT

The 'I' (insert) command is used to indicate where the moved ('M') or copied ('C') data is to be inserted. All lines previously saved by 'C', 'K', or 'M' commands will be inserted. To insert the moved or copied data at several points (also on different screens), simply specify multiple 'I' commands. The 'I' commands are processed after all other line commands; thus, you can move or copy data from the bottom to the top of the screen. (Functionally, the 'I' command causes a STACK CLOSE followed by a GETFILE $$STACK.)

Note: See note to C (copy) command.

K[nn]	STACK or DATA COLLECT

The 'K' (stack) command places 'nn' (1-99) lines, beginning with the line on which it is issued, into the stack area. This command is similar to the 'C' command except that the first 'K' command on a screen **does not** cause the stack to be opened. Thus, when collecting data from several screens to be copied to yet another location, the 'K' command should be used in place of the 'C' command on all screens except the first.

Note: See note to C (copy) command.

M[nn]	MOVE

The 'M' (move) command places 'nn' lines (1-99), beginning with the line on which it is issued, into the stack. You can later insert these lines into the same file, or another file using the I (insert) command. After being stacked, the lines are deleted from the file. Thus, the 'M' command is equivalent to a 'C' followed by a 'D' command. (Functionally, a 'C' or 'M' command causes a STACK OPEN followed by the placing of the lines in the stack. Compare this with the 'K' command above which does not cause the stack to be opened).

Note: See note to C (copy) command.

/	SET LINE POINTER

The '/' command positions the line pointer to the associated line. If multiple '/' commands occur within the same format area, the lowest one within the format area will take effect. Since line commands are processed before normal editor commands, the '/' command can be entered on a screen to set the line pointer to a line to be referenced by a normal editor command that is also to be entered on the same screen. The '/' command can be specified in the line command area preceding another line command.

"[nn]	DUPLICATE

The " (double quote) command duplicates (1-999 times) the line on which it is entered. Following the insertion, the cursor is set to the first of the duplicated lines.

```
>[nn[,mm]]    SHIFT RIGHT
<[nn[,mm]]    SHIFT LEFT
```

The '>' or '<' command shifts the data within the zone of the line on which it is issued to the right or to the left by 'nn' columns. Data shifted out of the zone is lost. If the second operand ('mm') is specified, the shift operation will be performed on 1 to 999 records beginning with the line on which the command is issued. The 'mm' operand cannot be specified if the NUMBERS option is on.

```
TA[nn]    TEXT ALIGN
```

The 'TA' command justifies the data within the zone left or right 'nn' lines, beginning with the line on which it is issued. (For more information see the description of the ALIGN command in this chapter).

```
TC[nn]   TEXT CENTER
```

The 'TC' command centers the data that is within the zone in the 'nn' lines beginning with the line on which the command is issued. (For more information see the description of the CENTER command in this chapter).

```
TL[nn]    TEXT LEFT JUSTIFY
TR[nn]    TEXT RIGHT JUSTIFY
```

The 'TL' or 'TR' command justifies the data within the zone left or right on the 'nn' lines beginning with the line on which it is issued. (For more information see the description of the JUSTIFY command in this chapter).

```
TS nn     TEXT SPLIT
```

The 'TS' command splits the line on which the command is issued into two lines. The 'nn' operand indicates the column number at which the split is to occur ($1 < nn < 80$). (For more information see the description of the SPLIT command in this chapter).

Chapter 5. Job Entry Statements

Job entry statements conform to the same syntactical rules as system commands: they begin with a slash (/) followed by an operation code and optionally by one or more operands separated by blanks. There are, however, important differences. System commands take effect as soon as they are entered, while job entry statements only take effect when the job is run, for example with a /RUN or /EXEC command.

In the following example, /INPUT is a system command which places your terminal into input mode so that you can enter the two job entry statements (/LOAD and /INCLUDE) into the input area. /ENDRUN is a command which terminates input mode and places the job in the input area into execution. Job entry statement /LOAD loads the VS FORTRAN compiler from a VSE library, job entry statement /INCLUDE includes some program out of member FPROG which is going to be compiled.

```
/INPUT
/LOAD VFORTRAN
/INCLUDE FPROG
/ENDRUN
```

Job entry statements can be stored together with programs and/or data in members, or they can be held separately in some extra member or in the input area and the related programs and data are included at execution time with the /INCLUDE job entry statement (see example).

Job Entry Statement Length

Job entry statements cannot extend past column 72. Only the /FILE and /LOAD job entry statements can be continued from one record to the next. To continue the /FILE statement, place a comma after the last operand in the first record. For the /LOAD statement, place a continuation character in column 72. Then start the continuation record with a slash and two blanks in columns 2 and 3. The continuation data can then begin in column 4.

Job Stream, Job Step

A job stream consists of one or more job steps. A job step starts with a /LOAD job entry statement and ends with the next /LOAD, or with end-of-job. A job step consists of job entry statements and user data. After a /LOAD statement has been processed, the first statement encountered that is not a valid job entry statement is assumed to be the start of user data for the job step.

A summary of all VSE/ICCF job entry statements appears on the next page together with a brief description of their functions. A detailed description of these statements appears on the pages immediately following this summary.

Summary of Job Entry Statements

STATEMENTS	FUNCTION
/ASSGN	Alters the default unit record device assignments for programs in interactive partitions.
/COMMENT	Serves as a comment within a sequence of job entry statements.
/DATA	Separates the input to a language compiler from data which the compiled program will read.
/FILE	Provides information about disk files which will be processed during an interactive partition execution.
/FORCE	Causes the lines in the print area to be displayed at the terminal.
/INCLUDE	Logically groups several library members and/or the contents of work areas into a single source of input.
/LOAD	Specifies the name of a program that is to be loaded and run.
/OPTion	Alters the standard setting of certain job processing options.
/PAUSE	Causes the scheduler to display a message (if 'comment' was specified) and then to stop processing the job stream.
/RESET	Causes any prior /ASSGN statements in the job stream to revert to the installation defaults.
/TYPE	Displays comments from a job stream.
/UPSI	Sets the VSE User Program Switch Indicators.

Figure 5-1. Summary of Job Entry Statements

/ASSGN Job Entry Statement

The /ASSGN job entry statement is used to alter the standard assignments for unit record devices used within programs in interactive partitions.

```
/ASSGN        SYSnnn    REAder|PUNch|PRInter|LOG|PUNCHIn|IGN|UA
```

SYSnnn is a decimal number from 000 to 240 indicating the VSE programmer logical unit to be assigned to a particular function. Your installation may have its own logical units, but here are the standard defaults:

Normal Default	Your Default	Device	Description
SYS005	________	READER	Job stream input device
SYS006	________	PRINTER	Printed (terminal) output
SYS007	________	PUNCH	Output to punch area
SYS008	________	PUNCHIN	Read from punch area
SYS009	________	LOG	Terminal input and output

If you are unsure which units to use for job stream input (READER), conversational input (LOG or READER with INCON), printed (terminal) output (PRINTER or LOG) or punch output (PUNCH), contact your VSE/ICCF administrator. See also *Chapter 9. Job Entry Considerations.*

To run a program that does not use standard default SYS units for unit record I/O, assign your SYS unit in your job stream. For example, to run a program that reads 80-character records from SYS042, specify /ASSGN SYS042, READER (unless SYS042 is the default for the job stream input).

Note that this assignment does not change the assignment of SYS042 established at VSE/ICCF startup time. It simply tells VSE/ICCF which I/O stream is to be intercepted. Thus, for the opening of the file to be successful, your assignment must be consistent with the one established at startup time. If the assignment is to a unit record device, the open will be successful. If it is to a DASD, you must supply a dummy /FILE statement, because OPEN checks for DASD labels. If it is not assigned, you cannot open the file.

IGN causes VSE/ICCF to ignore the default action for the specified SYS number. For example, if SYS005 is the programmer unit for job stream input, the statement /ASSGN SYS005, IGN will tell VSE/ICCF not to intercept read requests from SYS005; they should be allowed to pass through to the VSE/Advanced Functions supervisor for handling. IGN should only be issued for 'SYS' numbers that correspond to the installation defaults.

UA unassigns a previous SYS assignment. Once an assignment has been established for a SYS number, it will apply to all subsequent steps in the same job. To negate the effect of a previous SYS assignment and reassign it, you must unassign it by specifying UA and then assign it. To negate the effect of all prior assignments, issue the /RESET job entry statement.

Example:

The program (UTILX) to be run uses SYS004 for its control statement input and SYS005 for printer (terminal) output instead of the installation standard defaults of SYS005 and SYS006.

```
*READY
/input
/load utilx
/assgn sys004,reader
/assgn sys005,printer
     utilctl 75,300
/end
*READY
/run
*RUN REQUEST SCHEDULED FOR CLASS=A
```

/COMMENT Job Entry Statement

The /COMMENT statement is ignored. It can be used to put comments into a sequence of job entry statements.

```
/COMMENT [comment]
```

/DATA Job Entry Statement

The /DATA job entry statement is used to separate the input to a language compiler from any job stream data cards (SYSIPT or SYS005 or other installation default) which the compiled program expects to read.

```
/DATA           [INCon|NOIncon]
```

INCon	specifies that the program which has been set up to read data from SYSIPT or SYS005 (or other installation default for normal card read input) will instead ask for the input conversationally from your terminal.
NOIncon	can be entered while in INCON mode to conclude processing and revert back to job stream processing on SYSIPT.

Any data in the operand field other than INCON or NOINCON is treated as comments in the job stream.

If the program being compiled and run does not read input from the job stream VSE unit SYSIPT or the numbered SYS unit assigned to SYSIPT, there is no need to specify a /DATA statement.

A /DATA statement in a job stream invokes LINKNGO if the compiled object program has not yet been executed. That is, it has the same effect as specifying /LOAD LINKNGO. If no compilations have previously been performed, the /DATA statement forces end-of-file in the same way as a /* in a VSE job stream would.

When a /DATA statement implies the invocation of the LINKNGO program, it can be followed with job entry statements such as /FILE or /OPTION which are to apply to the load and go step.

The /DATA NOINCON statement can be entered while in conversational (INCON) mode to cause the running program to return to non INCON (SYSIPT) mode to continue reading data. This is useful if one program chains to another but the execution time input for the programs is to be done in conversational mode while the source program input is to be done in jobstream mode.

Example:

```
*READY
/input
7.4
5.9
/end
*READY
/save fortdta
*SAVED
*READY
/input
/load vfortran
5    read (1,10,end=99) a
10   format (f8.3)
     x=a**2
     write (3,15) a,x
15   format (' x=',f8.3,' x**2= ',f20.3)
     go to 5
99   stop
     end
/data
/include fortdta
/end
*READY
/run
*RUN REQUEST SCHEDULED
     . (fortran compiler and linkngo output)
X=       7.400 X**2=          54.760
X=       5.900 X**2=          34.810
****JOB TERMINATED - ICCF RC 00, EXEC RC 0000,
*READY
/ed
*EDIT MODE
find /d
/DATA
add  incon
P
/DATA INCON
next
/INCLUDE FORTDTA
*EOF
save fortprog
*SAVED
*READY
/exec fortprog
*RUN REQUEST SCHEDULED
     . (compiler and linkngo output)
?    . (question mark indicates a conversational read)
4.0
X=       4.000 X**2=          16.000
?
3.5
X=       3.500 X**2=          12.250
?
/*     (will cause end of file on fortran unit 1)
****JOB TERMINATED - ICCF RC 00, EXEC RC 0000,
*READY
```

/FILE Job Entry Statement

The /FILE job entry statement is used to provide information about disk files which will be processed in your interactive partition execution.

```
/FILE        param1 [param2 ... paramn]
```

param1...	are operands consisting of keywords followed by an equal sign followed by a parameter. The parameters define information about the file being specified.

File Statement Operands and Parameters:

BLKsize=n	This operand corresponds to the BLKSIZE parameter in the VSE DLBL job control statement. 'n' specifies the blocksize for 3350 and 3330 Model 11 SAM files. The operand is optional; if it is omitted, the blocksize value in the DTF is used.
BUFfer=bufsize	This operand corresponds to the BUFSP operand of the DLBL job control statement and applies only to VSE/VSAM files. It allows you to dynamically specify the number of bytes of virtual storage to be allocated for I/O areas for the specified file.
CATalog=catname	This operand only applies to VSE/VSAM files. It allows you to specify a file name (see the NAME= operand) of another file that is a VSE/VSAM catalog to be searched for the catalog entry for the file.
CISize=cinv	This operand corresponds to the CISIZE parameter in the VSE DLBL job control statement. 'cinv' specifies the control interval size for a SAM file on a FBA device. The control interval size must be a multiple of the physical blocksize of the FBA device. The CISIZE operand is optional and can be specified only in conjunction with TYPE=SEQ.
CYL=YES	The CYL operand allocates the number of tracks requested for dynamically allocated disk space in the SPACE operand to start and end on a full cylinder. The SPACE=ntrks parameter will be rounded to the next higher integral number of cylinders for the CKD device and ignored for the FBA device.

DATe=expinf	For normal VSE output files and dynamically allocated DISP=KEEP files you can specify the expiration information. If this operand is not specified, the file will not be date protected. You can specify a retention period as a number of days (e.g., DATE=14) or an absolute expiration date (e.g., DATE=81/235). The expiration date has either of three formats:
	`yy/ddd`
	`19yy/ddd`
	`20yy/ddd`

DISp= [delete]
DISp=[Pass]
DISp=[Keep]
	The disposition operand only applies to dynamically allocated files. If omitted, DELETE is assumed, which means that the file area should be released after the current job step has been completed.
	
	PASS indicates that the file should be retained (passed) from step to step and released at the end of the job in which it was allocated.

KEEP indicates that the file is to be permanent. It will be kept as long as the dynamic space area is 'warm' started or until you scratch it. If the dynamic space areas are 'cold' started, the file will be lost the next time VSE/ICCF is initiated. The file must be referred to as a normal VSE file in all jobs after the job which allocated it (i.e., it will require the LOC=operand).

IDent='fileident'	This operand can apply to dynamic space files or to normal VSE files. It is used to specify the file identification which identifies the file as it appears on disk. For normal VSE files, the default is the file name (NAME= operand). This parameter must be specified if the file identification is (input files) or will be (output files) different from the file name. For dynamic space files, the default for this parameter is 'filename.userid.termid'. 'userid' is the four-character user identification; 'termid' is the four-character terminal identification. 'filename' is the name specified in the NAME= operand, unless name is of the form IJSYS0n. If so, the J will be replaced with a K and the 0 will be replaced with the interactive partition identification character. The beginning and ending apostrophes can be omitted if the 'fileident' does not contain delimiter characters. You can also use the tags &USR, &TRM, and &PRT to form a unique 'fileident'. &USR will be replaced with your userid, &TRM will be replaced with the terminal identification and &PRT will be replaced with the interactive partition number. It is important to be able to specify standard job streams that reference file identifications based on a particular user. Or it might be convenient to base the standard job stream on a terminal, or on the interactive partition in which the job will be run. Thus, these tags can also be used in /PAUSE and /TYPE statements, so that the created file identification can be returned to you.
LOC=start,len	The LOCATION operand only applies to normal VSE disk files. For output files it must be specified. For input files it need only be specified if the installation is using the 'file protect' supervisor option. The 'start' parameter specifies the track or block (on FBA devices) number at which point the file begins on the disk. The 'len' parameter specifies the number of tracks or blocks (on FBA devices) in the file area.

MAXR=nnnnn
MAXR= <u>32767</u>
	Specifies the maximum number of records to be in the target member. MAXR can have a value from 1 to 99999, 32767 is the default. When this number is reached, the job is canceled (unless DTSPROCS is running) and the member is closed. At this point it will contain MAXR+1 records. All subsequent print or punch output will then be stored in either the print or the punch area. This keyword applies only to TYPE=ICCF output files.

NAMe=filename	The NAME operand must be specified on all /FILE statements. Except for TYPE=ICCF files, filename is the 1 to 7 character name used to define the file within the program being run. For TYPE=ICCF files, the parameter specifies the name of the VSE/ICCF library member for input or output. The member must exist within your library. When printing or punching to a library member, VSE/ICCF will extend the member if necessary. If UNIT=SYSLST or UNIT=SYSLLG has been specified, the member should be a print-type member. To create a print-type member, the suffix '.P' must be added to the filename, which must not exceed 8 characters, including the '.P'.

This suffix must be specified whenever a print-type member is referenced, except for $$PRINT, which is always considered to be a print-type member. All members of a generation member group must be either print-type or not print-type members. If a generation member group is of print type, the name part of the membername must not exceed 4 characters. The member type can only be changed via /RENAME, which sets the type to print or non-print, depending on whether the new name has the suffix .P or not. The period should not be used as a delimiter, backspace character or line end character. Control characters are placed into the records, regardless of the name you choose.

Note: The members specified in the NAME parameter must already exist.

PASsword=password	The PASSWORD operand should only be specified for TYPE=ICCF files which are password protected. Specify the four character member password.

RETAIN=JOB
RETAIN= STEP
	This keyword applies only to TYPE=ICCF punch or print files. It indicates how long a /FILE statement is to be effective. 'STEP' (default) applies only to the current job step. 'JOB' applies until end-of-job, until another /FILE statement, or until /OPTION PUCLOSE|PRCLOSE is encountered.
	
	Note: Using a member for different purposes in the same job can lead to unpredictable results.

SERial=serno	The SERIAL operand specifies a six-character volume serial number, which may be specified for normal VSE input and output files. For dynamic space allocation files, VSE/ICCF will attempt to find a dynamic space designated area on the specified volume from which to allocate the space. For DISP=KEEP files, if the space cannot be found on the specified volume, the job will be canceled. For temporary files (DISP= PASS or TEMP), if the space cannot be found on the volume, VSE/ICCF will look for space elsewhere.
SPAce=ntrks	The SPACE operand indicates a request for dynamic disk space allocation. Specify only the number of tracks or units of space (a unit of space on a FBA device is 16 physical blocks) required for your file. If you request more space than an installation specified maximum value, your request will be reduced to the maximum allowable value. *Note:* A certain amount of space (usually between one and two tracks) will be used by VSE/ICCF to store control information. This space is not available for allocation.

TYPe= Direct
TYPe= Seq
TYPe=Vsam
TYPe=Iccf
	For normal VSE files and for dynamically allocated file space, this operand specifies the file access type (DIRECT, SEQ, VSAM). SEQ is the default if the operand is not specified. For dynamically allocated file space, TYPE=VSAM should not be specified. For VSE/ICCF library members, TYPE=ICCF must be specified.

UNIt=sysno
UNIt=SYSLST
UNIt=SYSLLG
UNIt=SYSPCH

The UNIT operand can be coded to specify the 'SYS' number to be associated with the file. For dynamically allocated disk space files, VSE/ICCF will attempt to find a dynamic space designated area on the disk assigned to this SYS number. For DISP=KEEP files, if the space cannot be found on the specified unit, the job will be canceled. For temporary files, an attempt will be made to allocate the space from another area.

For normal VSE files, the 'SYS' number specified (any numbered unit or the system logical units) will be used to access the file so you should make certain that the 'SYS' number is assigned to the proper disk unit before using it. If the UNIT parameter is omitted, VSE/ICCF will use a 'SYS' number which it knows to be associated with the volume (SERIAL=) specified. In case a 'SYS' number cannot be found, the /FILE statement will be considered invalid.

For TYPE=ICCF input files, the UNIT parameter must be specified, and the 'SYS' number specified must be equal to the 'SYS' number in the CCB used to access the file and cannot be any system logical unit (i.e. SYSIPT). For TYPE=ICCF punch output to the library, the parameter must be specified as SYSPCH.

The UNIT=SYSLST, UNIT=SYSLLG and UNIT=SYSPCH operands concern only TYPE=ICCF members. UNIT=SYSLST specifies that only the print output produced for SYSLST is to be stored in the specified member. SYSLOG output is to be stored in the print area and displayed in the usual way. If UNIT=SYSLLG is specified, output for both SYSLST and SYSLOG is to be stored in the specified member. SYSLOG data is to be stored in the print area and displayed in the usual way. If the /FILE statement is used with UNIT=SYSLLG, but without TYPE=ICCF, UNIT=SYSLST is assumed. UNIT=SYSPCH is required for punch output to the VSE/ICCF library, but remember that RETAIN and MAXR also apply to SYSPCH.

Notes:

1. *The UNIT= parameter should* **always** *be specified for job streams which are submitted for execution in a batch partition. This ensures correct assignment for disk volumes which are not accessible from a VSE/ICCF interactive partition.*

2. *If UNIT=SYSLST or SYSLLG is specified, the NAME parameter should refer to a print-type member (xxxx.P) so that print format data can be obtained via the /LIST command.*

VOLume=n

The VOLUME operand only pertains to dynamically allocated files. It provides a method of distributing temporary space requests over multiple volumes for more efficient access. 'n' is a decimal number 0 through 9 corresponding to the 10 possible dynamic space areas within the environment (your installation may, in fact, only have 1, 2, or 3).

For temporary files, VSE/ICCF will attempt to allocate the file space on the specified volume regardless of serial number. If the space cannot be found, the space will be allocated from any available space.

File information need not be provided for a disk file if the file information for that file (DLBL/EXTENT statements) was present in the VSE/ICCF initialization job stream. A /FILE statement overrides the initialization job stream DLBL/EXTENT. The three major types of the /FILE statement request file information for:

1. Dynamically allocated sequential or direct files.

2. Normal VSE sequential, direct or VSE/VSAM files.

3. Sequential files that are members of your VSE/ICCF library.

These three major types are discussed below.

Specifying Dynamic Space Files:

If VSE/ICCF supports dynamic disk space allocation, you can ask the system to find space for your sequential or direct file dynamically. The basic request for file space would be:

```
    /FILE NAME=filename,SPACE=ntrks
```

For sequential input, output or work files, this is all that is required. If you want your file to have a name on disk different from the way the file is named in your program, add the IDENT parameter:

```
    /FILE NAME=filename,SPACE=ntrks,IDENT='fileident'
```

To make a file accessible from one job step to the next, you should also specify DISP=PASS. Otherwise the space will be made available for other users after the step in which it was allocated has been completed. If you want your file to be permanently accessible (from one job or from one day to the next), specify DISP=KEEP. If the file is not a sequential but a direct file, be sure to specify TYPE=DIRECT.

After a dynamic file has been allocated, you will receive a message telling you where the file was allocated (serial number, SYS number, relative disk address, etc.). For a DISP=KEEP file, note this information, because you will need it when you want to access the file later, when it will be specified as a normal VSE file.

Space on FBA disks is allocated by groups of physical records, which consist of 16 physical blocks and which are roughly equivalent to a track on a CKD disk device. Requests for disk space are therefore somewhat independent of the device type.

It is important to note for dynamically allocated files on FBA disk devices that the length field in the location information returned after allocation will be 16 times larger than the requested space. This need not concern you unless the file is DISP=KEEP. In this case note the location as it appears and use it in exactly the same way on subsequent /FILE statements for this file.

Specifying Normal VSE Files:

When running under VSE (not VSE/ICCF) you specify disk file information in DLBL/EXTENT statement sets. Under VSE/ICCF, this same information can be provided in a /FILE job entry statement. A more detailed description can be found in the corresponding VSE/Advanced Functions reference manuals.

When specifying disk file information for normal VSE input files, there are two or three required parameters:

/FILE NAME=xxxxxxx,IDENT='fileident',SER=serno

The IDENT parameter is only required if the file identification on disk is different from the file name which you use in your program.

Use the UNIT parameter if the logical unit for your file must be different from the logical unit of the volume on which your file resides.

Also if the file information is for an output file (a file which does not yet exist on disk), then you must specify the LOC parameter which specifies the relative disk address where the file is to be built. Moreover, if your installation uses the VSE file protect supervisor option, you must specify the LOC parameter for input files as well as for output files. Thus, you might have the following /FILE statement:

```
/FILE NAME=xxxxxxx,IDENT='fileident',SER=serno,
/     UNIT=sysno,LOC=start,len
```

If the file information was for an output file, and if you wanted the file to be date protected, you must specify the DATE parameter.

If the file information related to a direct or VSE/VSAM file rather than a sequential file, you must specify TYPE=DIRECT or TYPE=VSAM.

For a VSE/VSAM file you may also want to specify the CATALOG or the BUFSIZE parameters. The CATALOG parameter allows you to specify a catalog to be searched other than the master or user catalog. The BUFSIZE parameter allows you to dynamically specify main storage space to be allocated to the file for I/O buffers.

Specifying Special VSE/ICCF Member Files:

Normally, all punch directed output from a program is placed in the punch area. However, you can also have the punch output from a job step placed directly into a library member. The member must be in the primary library and you must specify the following /FILE statement:

```
/FILE NAME=member,UNIT=SYSPCH,TYPE=ICCF
```

UNIT=SYSPCH must be specified regardless of what SYS unit actually produces the punch output. If the member is password protected, you must specify the PASSWORD parameter.

It is always possible to read a library member into a program by having an /INCLUDE for the member in the job stream. Sometimes, however, it may be necessary to read more than one member into the program simultaneously. To do so, you can specify:

```
/FILE NAME=member,TYPE=ICCF,UNIT=sysno
```

The sysno parameter must match the SYS number in the CCB that is used to read the file. The capability can only be used by programs which are specially written to read records at the EXCP level without doing a logical IOCS open for the card read file.

The number of VSE/ICCF-type input files a program can use is restricted to two.

You can also save the total print output from a job or a jobstep in the VSE/ICCF library up to the value specified in MAXR. This allows you to obtain a hard copy of the output by simply sending the member to the printer using the RELIST macro. The format of the records in such a member corresponds to the format of the records in your print area. Exceptions are those special records which indicate conversational read requests or full screen write/read requests. These records are replaced in the member by the messages:

*CONVERSATIONAL READ or * FULL SCREEN WRITE/READ.

Examples:

1. Specify work files for an assembly of the program called ASMB. Place the object deck (punch output) into a preallocated member called ASMBOBJ.

```
*READY
/input
/load assembly
/file name=ijsys01,space=10
/file name=ijsys02,space=10
/file name=ijsys03,space=10
/file name=asmbobj,type=iccf,unit=syspch
/option noload,deck,nolist
/include asmb
/endrun
*RUN REQUEST SCHEDULED FOR CLASS=a
```

2. Specify file information to access a VSE library.

```
*READY
/input
/load libr
/file name=ijsyssl,id='source.lib',unit=sys070,
/  serial=lib008
   access subl=ll.test
   listdir fregs.a
/endrun
*RUN REQUEST SCHEDULED FOR CLASS=A
```

Note: If serial= is not specified, the symbolic unit (SYS070 in the example) must be assigned to the VSE/ICCF partition.

3. Create a direct file named SPECLIB and date protect it.

```
/input
/load specbld
/file name=speclib,ident='wksp.lib',ser=222222,
/  loc=2400,200,date=99/365,type=direct
/include specdata
/endrun
```

4. Dynamically allocate a file for later use and then clear and format the file area (this example is not valid for FBA devices).

```
/input
/load clrdk
/file name=uout,id='jc.fort.data',space=20
// ucl B=(K=0,D=100),ON,E=(3340)
// end
/endrun
```

5. Dynamically allocate a file (using DYNAMIC Space on FBA) for later use with DATE PROTECTION, merge data from two card type files, write data to the file (program BLDFILE), and read it back to create a listing in a later job.

```
/inp
/load bldfile
/file name=indata,type=iccf,unit=sys005
/file name=outfile,id=extract,date=10,space=100,volume=3
/include mstrdata
/endr
* RUN REQUEST SCHEDULED
K859I...ALLOCATION FOR OUTFILE -SERIAL=WORKDS UNIT=SYS002
        LOC=3200,1600
 .
 .  (results of file built from program)
 .
* *** JOB TERMINATED RETURN CODE 00 NORMAL EOJ
* READY
/inp
/load prntfile
/file name=inprint,id=extract,loc=3200,1600,
/  SER=WORKDS,UNIT=SYS002
/endr
* RUN REQUEST SCHEDULED
 .
 .  (job completes)
 .
```

/FORCE Job Entry Statement

The /FORCE job entry statement forces the lines in the print area to be sent to the terminal as soon as the next print activity occurs.

```
/FORCE
```

Normally a job in execution will not begin displaying data until the job ends, or until a conversational read is encountered, or until the print (spool) area is full. Sometimes, however, you may need to force output to the terminal before the print area is full. The /FORCE job entry statement allows you to do this.

The /FORCE statement can be imbedded within data records, or it can be included with other job entry statements.

Example:

```
*READY
/input
/load vfortran
/include primegen
/load loader
/load vfortran
/force                     (Will cause previous output to be
/include fibonaci          directed to terminal before second
/endrun                    compile is run.)
*RUN REQUEST SCHEDULED
```

/INCLUDE Job Entry Statement

The /INCLUDE job entry statement logically groups library members and/or the contents of work areas into a single source of input.

```
/INCLUDE     name [password][ICCFSLI]
```

name	is the name of a library member that is to be logically included in the job at execution time. The inclusion occurs at the point where the /INCLUDE statement appears. $$PUNCH, $$PRINT or $$LOG can be specified to include the contents of these areas.
password	is a four-character password and need only be specified if the member being included is password protected.
ICCFSLI	applies only to a job stream that is submitted to VSE/POWER via the SUBMIT procedure. The ICCFSLI parameter causes the /INCLUDE statement to be transformed into the VSE/POWER JECL statement * $$ SLI. A more detailed explanation follows on the next page, behind the example.

When building a job stream it is often inconvenient to have the program, data and job entry statements all together in the input area, or in a single library member. The /INCLUDE statement makes it possible to separate some or all job entry statements from the programs and/or data to which they refer. An /INCLUDE statement in the job stream has exactly the same effect as if all of the included data were physically located at the point of the /INCLUDE.

An /INCLUDE statement must be within a job stream and cannot be entered into a job stream via a /PAUSE statement.

/INCLUDE references can be nested to eight levels. The /INCLUDE statements can reference library members which themselves contain /INCLUDE statements. A member can chain to the next member by having an /INCLUDE as its last line. Up to 256 library members can be chained together in this way.

Normally you would enter a VS FORTRAN compile and run with the following job entry statements and data.

```
/LOAD VFORTRAN
 .   (FORTRAN Source Program)
/DATA
 . (Input Data)
```

However, if the data was in a separate member of the library named FORTDATA, the job stream might look as follows:

```
/LOAD VFORTRAN
 .   (FORTRAN Source Program)
/DATA
/INCLUDE FORTDATA
```

Or, the program itself can be in yet another member named FORTPROG. The job entry statements would then look as follows:

```
/LOAD VFORTRAN
/INCLUDE FORTPROG
/DATA
/INCLUDE FORTDATA
```

Now if the FORTRAN program was large and was stored in three library members named FORTPR1, FORTPR2 and FORTPR3, the job stream would appear as follows:

```
/LOAD VFORTRAN
/INCLUDE FORTPR1
/INCLUDE FORTPR2
/INCLUDE FORTPR3
/DATA
/INCLUDE FORTDATA
```

If the /INCLUDE statement in a SUBMIT job stream has no ICCFSLI parameter, the member will be read from the VSE/ICCF library file, then written to the VSE/POWER reader queue, and finally read from the reader queue into the program. If, however, the ICCFSLI parameter is specified, the SUBMIT program transforms the /INCLUDE into an * $$ SLI statement. The data of the member to be included will not get into the reader queue. Rather, VSE/POWER will read the member directly from the VSE/ICCF library file prior to execution time. This improves the response time from the SUBMIT procedure and also diminishes the channel utilization due to reduced data transfer.

Likewise, if the /INCLUDE statement has the ICCFSLI parameter specified, /INCLUDE statements within the member to be included (nested /INCLUDEs) are later resolved by VSE/POWER, regardless of whether they have the ICCFSLI parameter or not. The number of nesting levels is unlimited. /INCLUDE statements in compressed members as well as in noncompressed members will be resolved.

The * $$ SLI statement which the SUBMIT program generates from the /INCLUDE has the following format:

* $$ SLI ICCF=(member[,password]),LIB=(n1[,n2[,n3]])

'member[,password]' indicates the name of the member to be included, together with its password if it is password protected. LIB=(n1,n2,n3) specifies the numbers of the VSE/ICCF libraries to be searched. The VSE/ICCF library numbers reflect the terminal user's search chain at the time when the submit was issued, that is, primary library, connected library, common library.

Notes:

1. *The /INCLUDE with the ICCFSLI parameter cannot be used if the job stream is submitted to a remote node, unless the member exists at the remote system under the same member name and in the same VSE/ICCF library number.*

2. *A job stream containing * $$ SLI statements with the ICCF parameter can be entered from a card reader.*

3. *The VSE/ICCF library file is not available for the ICCFSLI function whenever one of the library modification processes of the DTSANALS or DTSUTIL utilities is active. It is possible that a submitted job requesting an SLI inclusion is canceled due to a library modification currently in progress.*

4. *A * $$ JOB and/or a * $$ EOJ statement in a member that is included with the ICCFSLI option is ignored.*

Example:

```
*READY
/input
card 1
card 2
/end
*READY
/save group1
*SAVED
*READY
/input
line 1
line 2
/end
*READY
/save group2
*SAVED
*READY
/input
/load object       (This program reads cards and
/option nogo        transfers them to the terminal.)
/upsi 1
/include group1
/include group2
/endrun
*RUN REQUEST SCHEDULED
CARD 1
CARD 2
LINE 1
LINE 2
****JOB TERMINATED - ICCF RC 00, EXEC RC 0000, NORMAL EOJ
```

/LOAD Job Entry Statement

The /LOAD job entry statement is used to specify the name of a language processor (assembler, compiler), a utility program or some other program that is to be loaded and run.

```
/LOAD        phasename [PARM='parameter']
```

phasename	is a 1 to 8-character name representing a phase that is cataloged in a VSE library.
PARM='parameter'	allows a parameter of up to 100 characters to be passed to the loaded program. If the parameter does not fit into the first 71 positions of the 80-byte record, a continuation record can be used. Put a continuation character into column 72, and start the continuation record with a '/' followed by 2 blanks.

The parameter is passed to the loaded program at invocation via register 1. It points to a pointer field that contains a 1-byte parameter status switch and a 3-byte pointer to the parameter field. The switch is:

X'80'	Parameter value available
X'00'	Parameter value not available

A 3-byte pointer points to the parameter field if it exists, or it is zero. The value field starts with a two-byte length field for the parameter value followed by the 1 to 100 byte parameter value.

The phasename specified is the name under which the program is cataloged in a VSE library (with the exception of ICDDSBSC which is specified as BASIC).

You will be prevented from using certain VSE library programs unless your user profile indicates that you are an authorized user.

The /LOAD statement should be the first statement in any job step. Any lines in the job stream prior to the first /LOAD statement will be ignored. Likewise, if a step of a multistep job terminates before reading all of its data, any lines in the job up to the next /LOAD statement will be ignored.

The phase name on the /LOAD statement will often be one of the following:

ASSEMBLY	specifies that the following lines are to be processed by the assembler program and that the resulting object program is to be run unless otherwise specified.
BASIC	specifies that the following lines are to be processed by the VS BASIC compiler and that the resulting program is to be run unless otherwise specified.
FCOBOL	specifies that the following lines are to be processed by the DOS/VS FULL ANS COBOL compiler and that the resulting object program is to be run unless otherwise specified.
SCOBOL	specifies that the following lines are to be processed by the DOS SUBSET ANS COBOL compiler and that the resulting object program is to be run unless otherwise specified.

VFORTRAN	specifies that the following lines are to be processed by the VS FORTRAN compiler and that the resulting object program is to be run unless otherwise specified.
PLIOPT	specifies that the following lines are to be processed by the DOS PL/I Optimizing compiler and that the resulting object program is to be run unless otherwise specified.
RPGII	specifies that the following lines are to be processed by the DOS/VS RPG II Compiler and that the resulting object program is to be run unless otherwise specified.
OBJECT	specifies that the lines following in the job stream are an object deck (not BASIC) which are to be written into the punch area and run. OBJECT will also transfer lines to the print area if an UPSI switch is set on. An execution of OBJECT is implied if an object deck is found in the input job stream and the deck is not preceded by a /LOAD. For further information on this option see the description of the OBJECT utility in *Chapter 8. Utility Programs.*
LINKNGO	specifies that the contents of the punch area are to be loaded into storage for execution. In most situations, /LOAD LINKNGO need not be specified since its use is often implied (see the description of LINKNGO in *Chapter 8. Utility Programs*).
DTSPROCS	specifies that the lines following in the job stream are to be read and interpreted by the Procedure Processor (DTSPROCS). For more information on using procedures, see *Chapter 7. Writing Procedures and Macros.*

Notes:

1. *Considerations for specific compilers are described in Chapter 9. Job Entry Considerations.*

2. *It is possible to load together (via LINKNGO) object programs created by all of the above compilers except BASIC. When loading programs and subprograms, the standard VSE linkage conventions apply. These conventions are usually given in the programmer's guide associated with the programming language.*

3. *Requests for any of the above language compilers can be grouped together in a single job.*

Example:

```
*READY
/input
/load assembly
/tabset asm
/set tab=;
main;start;0
;balr;5,0
;using;*,5
;excp;conccb
;wait;conccb
;eoj
conccb;ccb;syslog,conccw
conccw;ccw;9,condta,0,l'condta
condta;dc;c' this means the program worked'
;end
/endrun
*RUN REQUEST SCHEDULED
(assembler and LINKNGO output)
THIS MEANS THE PROGRAM WORKED
**** JOB TERMINATED - ICCF RC 00, EXEC RC 0000, NORMAL EOJ
*READY
```

/OPTION Job Entry Statement

The /OPTION job entry statement is used to alter the standard setting of certain job processing options.

```
/OPTion        option1 [option2 ... option n]
```

option1 ...n	are options which are to be set for a particular job step or series of steps. As many options can be specified on one statement as will fit into 72 columns. The options must be separated by at least one delimiter. Multiple /OPTION statements can be included in a single job step.

The options should not extend past column 72 and they should be separated by at least one space or a comma. If more options are required than can be specified on one line, additional /OPTION statements can be used.

Once an option is set for a job, it usually remains in effect until it is reset by another /OPTION statement. All options are always reset to their default status before the first step of each job.

Certain options apply to the general operation of VSE/ICCF, while others apply to individual language processors. If the information concerning the language processor options given below is not sufficient, you should refer to the Programmer's Guide for the language processor concerned.

BASIC, VS FORTRAN, and PLIOPT compilers do not check the standard /OPTION statement for language processor options. Instead, all BASIC options are specified on the BASIC RUN statement, all PLIOPT compiler options are specified on the *PROCESS statement, and all VS FORTRAN compiler options are either specified in the @PROCESS statement or in the PARM field of the /LOAD statement. Also, certain of the other language compilers have their own supplementary option statements; for example, the ANS COBOL 'CBL' statement.

The following are the language processor options which can be specified on the /OPTION statement. The standard default options are underlined. See the appropriate programmer's guides for the exact meaning of the option. Once again, these options do not apply to the VS BASIC, not to the PL/I, and not to the VS FORTRAN compilers.

NOALIGN ALIGN	is used by the assembler to control whether or not halfword, fullword and doubleword constants and storage areas are to be aligned on their appropriate boundaries.
DECK NODECK	causes an object program to be placed in the punch area from which it can be read in and run by the LINKNGO program. Specifying NODECK will make the compiler or assembler run faster but execution will not be possible.
NOEDECK EDECK	is only used by the assembler program. When the option is specified, the VSE assembler produces edited macro decks for any macros included in the assembly and places these decks in the punch area.
NOLIST LIST	is used by some compilers to control whether a listing of the input source program is produced on SYSLST (that is, on the terminal). NOLIST is the default for all compilers, since a listing of the source has usually already been made using the /DISPLAY or /LIST commands.

<u>NOLISTX</u> LISTX	is used by certain language compilers to control whether a hexadecimal representation of the generated object program is produced on the terminal. NOLISTX is the default since this type of output is usually lengthy and of questionable value. It is not used by the assembler or RPG II.
<u>SUBLIB=AE</u> SUBLIB=DF	is used by the assembler to control the source statement sublibraries which are accessed for COPY statements and macro definitions.
<u>NOSYM</u> SYM	is used by certain language compilers to control whether or not a printout of symbolic names defined in the program will be produced on the terminal. This option is not used by the assembler or RPGII.
<u>NOXREF</u> XREF	is used by the assembler and some language compilers to determine whether or not a cross-reference listing of symbolic names defined within the program is to be produced on the terminal.
RLD NORLD	is used by the assembler to control whether or not the relocation dictionary is printed at the end of an assembly. The default is the value set within your installation.

The following options apply to the control of jobs within the interactive environment. These options are unique to VSE/ICCF or have unique meanings within VSE/ICCF.

ANYPHASE=nn	Normally the COMREG (communication region) field which defines the high end of any phase with the same first four characters in its name as the initially loaded phase is set to the end of the first phase loaded. If this address proves unsatisfactory, you can set this value to any arbitrary point within the non-GETVIS portion of the interactive partition. For example, ANYPHASE=40 will set this address to the 40K point within the interactive partition.
<u>CLEAR</u> NOCLEAR	Unless NOCLEAR is specified, the entire interactive partition address space (except for the first 6K) will be cleared whenever a job ends. If for some reason you want the area to be retained, specify the NOCLEAR option.
<u>NOCONT</u> CONTINUE	The CONTINUE option causes the job to be placed in continuous output mode just as if the /CONTINU command had been entered after the /RUN or /EXEC. In continuous mode, print output is displayed continuously; you need not press ENTER, or the carriage return key, to obtain each new screen of data.
<u>NODUMP</u> DUMP	When the DUMP option is set, an interactive partition abnormal termination will cause the DUMP program to be invoked. The DUMP program will then allow you to display registers and storage areas within your program. The DUMP program is invoked for VSE/ICCF termination codes 02, 07, 08 and 14. The dump program is placed in the SVA; a work area of 3K is reserved in the interactive partitions's GETVIS area; if the address of this area is destroyed, the first 3K block within the interactive partition is taken as work area.
<u>EOFPRT</u> NOEOFPRT	When the NOEOFPRT option is set, no end of file record (/* */) will be placed in the print area at the end of the print output after the job has terminated. This option may be useful, for example, in handling conversational reads, which reset the pointer to the start of the print area. In this case, the end-of-file record would be written at the start of the print area, thus effectively destroying the contents of the area.

GETVIS=nnn
GETVIS=P-nnn
GETVIS=AUTO

Normally in VSE/ICCF, 48K of your background storage area is reserved as GETVIS space. You can alter this default for a job step by specifying the actual amount of storage to be reserved as GETVIS area. 'nnn' is the amount of GETVIS area to be reserved in increments of 1024 bytes. Depending on the environment in which VSE/ICCF runs, the amount which you specify will be rounded up to the next multiple of the page size. GETVIS=64 would reserve 64K (65,535) bytes. GETVIS=0K would indicate that a minimum GETVIS space of 48K was to be reserved. You can also specify the GETVIS area by specifying the entire partition minus some amount (P-nnn). For example, if your program required 52K, you might specify GETVIS=P-52. This would reserve 52K for your program (actually, slightly less than 52K due to some VSE/ICCF save areas at the low end of the interactive partition). The rest of the interactive partition would be allocated to the GETVIS area. GETVIS=AUTO tells VSE/ICCF to calculate GETVIS space according to the largest phase in the program to be loaded. VSE/ICCF takes the phase with the highest ending address that starts with the same four characters as the program to be loaded. The remaining space in the interactive partition becomes GETVIS space, provided minimum and maximum requirements are met. The minimum value for the GETVIS area is 48K. Requests for less than this will be set to 48K without any error messages. The GETVIS area can never exceed partition size minus 20K, and is reset to 48K at the start of every job step.

GO
NOGO

This option controls the automatic invocation of the LINKNGO program. Normally, it is invoked automatically at the end of a job when a compiler (except BASIC) has produced an object module which has not yet been run or whenever a /DATA job entry statement is encountered following the input to a compiler (except BASIC). However, when the NOGO option is on, LINKNGO will never be implicitly invoked. (This does not preclude explicit invocation.) This option can be used if you want to only produce the object program (but not load and run it). For example, you may only want to save it in the library. The options LOAD and NOLOAD will be accepted to provide compatibility with ETSS II; they have the same meaning as GO and NOGO.

NOINCON
INCON

This option controls where card read data, VSE units SYSIPT and SYS005 (or other default unit), is obtained. Normally, any card read data is obtained from the job stream itself. It is processed sequentially until all data has been read and processed. When the INCON option is specified, any input to be read from VSE units SYSIPT or SYS005 (or other default unit) will be directed instead to the terminal for conversational input. This is one way of writing a conversational program in programming languages which do not support input from the console (SYSLOG or SYS005). This option has the same effect on programs run by the /LOAD function as the /DATA INCON option has on programs loaded by LINKNGO from object modules.

NOJSDATA
JSDATA

Normally, a /LOAD job entry statement signifies the end of one job step and the beginning of the next job step. Also a /DATA statement signals end of file to the job reading data from the job stream. Occasionally, however, it may be useful to be able to read actual VSE/ICCF job streams into a program. This would require that the checks for /LOAD and /DATA statements be bypassed. The JSDATA (Job Stream as Data) option causes such a bypass. All remaining 80-character records in the job stream (beyond the first set of job entry statements) can be read into the running program without interpretation by VSE/ICCF. Note, however, that the /INCLUDE statement will still be interpreted normally, regardless of the setting of this option.

NOLOG LOG	When the LOG option is specified, all job entry statements encountered by the job scheduler will be displayed. This may be useful when you are initially debugging job streams.
NOOBJECT OBJECT	Normally, when VSE/ICCF encounters an object deck in a job stream, it forces end-of-file for the previous step's data input and then loads the object deck for execution as the next step of the job. However, you may sometimes want to process an object deck as ordinary data, for example when running the VSE Librarian to catalog an object deck. By specifying the OBJECT option, the object deck in the job stream is treated as ordinary data.
NOPERM PERMFILE	Normally, an OPEN request for a file named IJSYS0n will be converted to a request for IKSYSpn where 'p' is the interactive partition identifier. Sometimes, however, you may want to bypass this processing and actually open the file under the IJSYS0n name. This often occurs in FORTRAN when permanent files must have the IJSYS0n name. When the PERMFILE option is set, the file name will not be altered and the OPEN will proceed for the unaltered file name. The PERMFILE option does not apply to compilation. It only applies to the execution of programs other than compilers.
PROMPT NOPROMPT	Normally, when a conversational read is encountered, VSE/ICCF will cause a prompt to be issued to the terminal. For typewriter terminals the prompt is a question mark. For 3270 terminals the prompt is '*ENTER DATA'. When NOPROMPT is specified, the issuance of the prompt is bypassed. This facility can be useful if the program being run issues its own prompt.
SAVE NOSAVE NORESET RESET	The SAVE and RESET options together control the setting of the punch area EOF pointer and current (or next) record location pointer. Normally, at the start of a job, the punch area is set to a null condition. This means that its pointers are reset and it is reused from its beginning. Each step that places data in the punch area places this data immediately after the punch area from the previous step. In this way, for example, successive compilations can create several subroutines which are to be linked together via the LINKNGO program. After a request for the LINKNGO program has been processed, the line pointer is set back to the beginning of the punch area for later compilations and executions. When the SAVE option is used in the first step of a job it causes the automatic 'set to null' function (which is usually performed during the initiation of the initial job step) to be bypassed. Thus, the punch area output of a previous job (not step) can be made available to the succeeding job if the first step of the succeeding job has the SAVE option set on. This is useful, for example, if you want to access the object deck from a compilation performed in a previous job. The SAVE option is automatically set when the DTSPROCS utility is run. The RESET option resets the record pointer (in the punch output area) to the first record in the punch area. Normally, the next record pointer in the punch area will be set to the next output record past the output of the previous step within the job. When the RESET option is on, the pointer will be set back to the first record within the punch area. This option is useful, for example, if you want to reset the pointer to the beginning of the punch area since you have finished processing the existing data within the punch area.

NOSPECIAL SPECIAL	The SPECIAL option must be set if the job to be run will use any special programming techniques such as rewriting job stream data to disk (read-no-feed followed by write) or reading backwards in the job stream. If this option is used, however, it is your responsibility to check for special conditions, such as end-of-data when reading backward.
TIME=mm[,nn]	This option may be used to set the execution time monitoring factors for the job. 'mm' is specified in execution units which closely approximate seconds of elapsed (wall-clock) time. If the job exceeds this value it will be automatically canceled. 'nn' is specified in seconds. It represents the total amount of time the job may be in the interactive partition whether running or not. The maximum values for 'mm' and 'nn' are 32767 and 65535. This option provides the same facility within the job streams as the /SETIME command provides on an immediate basis.
NOTRUNC TRUNC=nn	This option causes printer output to be truncated from the normal print line width to 78 characters. This is useful where the line width of the physical terminal is not wide enough to accommodate the print line. The use of the TRUNC option makes the output easier to read. When TRUNC is not in effect, any lines whose width exceeds the capacity of the device will be displayed on two lines. If 'nn' is not specified or is specified as '00', the first 78 characters of the print line will be written to the print area. When 'nn' is specified, 'nn' columns are removed from the front of the print line and the next 78 bytes are written to the print area. Thus, TRUNC=20 would cause print positions 21 through 98 to be displayed. The TRUNC option can also cause a job to run faster since less data need be written to the print area and to the terminal.
PRCLOSE PUCLOSE	The PRCLOSE option closes a member specified in a previously encountered /FILE statement that contained UNIT=SYSLST or UNIT=SYSLLG, and directs further print output to the print area. Further output will be placed after any records already in the print area. PUCLOSE has the same effect for a /FILE statement that contained UNIT=SYSPCH, and directs further punch output to the punch area.
SYSLOG SYSIPT	This option allows you to let the VSE librarian read its input from either SYSLOG or SYSIPT. When the VSE librarian runs in a batch partition, it reads from SYSIPT or SYSLOG depending on the VSE job control statements by which it is invoked. When running in an interactive partition, the VSE librarian takes its direction from the SYSLOG/SYSIPT option. SYSLOG means that the librarian will prompt you for its input. SYSIPT causes the librarian to read its input from SYSIPT. This option can also be used for other programs that optionally read from SYSLOG or SYSIPT.

Example:

```
*READY
/input
/load vfortran
/option list listx xref trunc
/option getvis=auto
     write (3,10)
10   format ('VSE/ICCF demo')
     end
/end
*READY
/run
*RUN REQUEST SCHEDULED
```

/PAUSE Job Entry Statement

The /PAUSE job entry statement is used in a job stream to cause the scheduler to display a message (if 'comment' is specified) and then to halt.

```
/PAUSE        [comment]
```

comment   is a message which will be displayed on the terminal at the time the pause occurs.

During the halt you can type in job entry statements to further define the job or merely press the ENTER or EOB key to cause the scheduler to continue processing. /INCLUDE statements cannot be entered at a /PAUSE but must be within the job stream.

The tags &USR, &TRM, or &PRT can be used in the comments. '&USR' will be replaced with your userid, '&TRM' will be replaced with the terminal identification, and '&PRT' will be replaced with the interactive partition number before the message is written to the terminal.

Example:

```
*READY
/input
/load spcutil
/file name=utlwork,tracks=10
/pause ** enter appropriate upsi **
/endrun
*RUN REQUEST SCHEDULED FOR CLASS=A
*     - ALLOCATION FOR UTLWORK - SERIAL=111111 UNIT=SYS001
        LOC=3267,10
**ENTER APPROPRIATE UPSI**
?
/upsi 1011
?
(eob or CR)
 .
 . (job continues)
 .
```

/RESET Job Entry Statement

The /RESET job entry statement is used in a job stream to cause any prior /ASSGN statements to revert to the installation defaults.

```
/RESET
```

/TYPE Job Entry Statement

The /TYPE job entry statement is used to have comments displayed.

```
/TYPE         comment
```

comment is a line of comment data that you want displayed during the execution of the job in which it occurs. The comment string may contain double-byte characters.

The comment will appear on your screen at the point in the job stream at which it occurs.

The tags &USR, &TRM, or &PRT can be used in the comments. &USR will be replaced with your userid, &TRM will be replaced with the terminal identification, and &PRT will be replaced with the interactive partition number.

Example:

```
*READY
/input
/load listutil
/file name=savdta,ser=etspak,id='jc.saved.data'
/type **********
/type *the following is a
/type *listing of the file
/type **********
list file=savdta
/endrun
*RUN REQUEST SCHEDULED
**********
*THE FOLLOWING IS A
*LISTING OF THE FILE
**********
ABRAMS, JOHN 234 ANY STREET
BAKER, B.A. 1912 FOREST DR
.
. (etc.)
.
```

/UPSI Job Entry Statement

The /UPSI job entry statement is used to set the VSE User Program Switch Indicators from the job stream.

```
/UPSI        string
```

string is a string of 1 to 8 zeros (0), ones (1) or X's. A zero sets the corresponding switch off, a one sets it on and an 'X' leaves it at its previous setting.

If a position is not coded, an 'X' is assumed.

Under VSE there are eight User Program Switch Indicators (UPSI) which can be set in the job stream and tested by the program.

All UPSI switches are set off (zero) at the beginning of each job. Once an UPSI switch is set on during a job, it remains on until it is turned off by another /UPSI statement.

The UPSI switches are generally referred to as UPSI-0, UPSI-1...UPSI-7. The switches can be referred to in assembler language via the COMRG macro, in ANS COBOL via SPECIAL-NAMES, and in RPGII by referring to indicators U1 through U8. The FORTRAN and PL/I user can refer to UPSI switches via an assembler subroutine.

Programs which will be compiled, loaded (by the LINKNGO utility) and run in interactive partitions should not use UPSI-0. This UPSI is used in LINKNGO for control of where to read the object deck.

Examples:

1. Set UPSI-0 switch off and UPSI-1 and UPSI-2 on. Leave switches UPSI-3 through UPSI-7 at their previous settings:

   ```
   /UPSI 011
   ```

2. Set UPSI-3 switch on, leave all other settings at their previous value:

   ```
   /UPSI XXX1
   ```

3. Check UPSI setting in an assembler program and set UPSI at program invocation:

   ```
       *READY
       /input
       /load     assembly
       upsame    start       0
                 print       nogen
                 balr        5,0
                 using       *,5
                 comrg
                 mvc         msg+10(3),=c'off'
                 tm          23(1),X'40'         test upsi-1
                 bz          swoff
                 mvc         msg+10(3),=c'on '
       swoff     excp        msgccb
                 wait        msgccb
                 eoj
       msgccb    ccb         syslog,msgccw
       msgccw    ccw         9,msg,0,l'msg
       msg       dc          c'switch is xxx'
                 end
       /load linkngo
       /upsi 01  (this will set on UPSI-1)
       /endrun
       *RUN REQUEST SCHEDULED
       (assembler, LINKNGO and execution output)
   ```

Chapter 6. Dump Commands

This chapter defines each dump command, gives its format and usage and provides an example illustrating its use.

When Can a Dump Command be Entered?

If the DUMP option (/OPTION DUMP) has been specified in a job stream and the interactive partition execution terminates due to an invalid condition, program check or VSE/ICCF return code 02, 07, 08 or 14, the dump program will be automatically invoked. When the message:

K404D ENTER DUMP COMMAND

appears, any of the dump commands described in this chapter can be entered. System commands, which are also effective when a conversational read is outstanding (such as /SKIP and /CANCEL), are also effective at this time.

The Null Command

If ENTER (or EOB or Carriage Return) is pressed without entering a command, a display forward dump command is assumed as follows:

```
DISPFWD * +256
```

This command displays the first or next 256 bytes of your program storage in hexadecimal and character format. Pressing ENTER again will cause the next 256 bytes to be displayed, etc.

The Scan/Locate Pointer

Certain dump commands are designed to set what is called the scan/locate pointer. This pointer is initially set at the program load point and can be varied by the SEARCH, LOCATE, TOP, POINT, FORWARD, BACKWARD and DISPFWD dump commands. The address value of the scan/locate pointer can be referenced in dump commands by the character '*'.

For example, to locate and display data areas or constants which are in static storage locations (COBOL WORKING-STORAGE) you might precede certain groups of key constants or data areas with a unique identifier as in the following example:

```
01 CONST-AREA.
03 DBG-IDENT PIC X(7) VALUE  '**DBG**'.
03 PASSCODE  PIC X    VALUE  '0'.
03 TOT-FIELD PIC S9(7) VALUE +0.
03 COUNT-FLD PIC X9(5) VALUE +0.
```

Then if your program abnormally terminates, you might issue the following commands to locate and display the area.

```
    K404D ENTER DUMP COMMAND
    locate '**DBG**'
    403756 0008CE 0008CE   5C5CC4C2C75C5C ...    **DBG** ...
    K404D ENTER DUMP COMMAND
    dispfwd * +128
    ... (storage display)
```

Reference Area/Origin Command

Whenever a dump command calls for an address operand, it is possible to specify a hexadecimal relative address. Usually the address which you specify is obtained from a compiler listing and thus will normally be relative to the program's load point. This is the assumption made by the Dump Program. However, it is possible to vary the reference area by using the ORIGIN dump command.

The ORIGIN command allows you to vary the base address for relative address calculation. This can be useful when displaying fields within an area when all you have is a listing of fields relative to some base point.

This can also be useful when you are debugging a subroutine or sub-program loaded with the main program. By setting the ORIGIN to the start of the routine, all relative addresses would then correspond to the program listing for the subroutine.

Another use might be to set the origin or reference area to the start of the main program area when this area is for some reason not the first control section loaded; in other words, when the main portion of the program does not start at the program load point.

Summary of Dump Commands

COMMAND	FUNCTION
ADD	Adds two hexadecimal numbers or one hexadecimal number and the contents of a general purpose register.
Backward	Reduces the scan/locate pointer by a given number of bytes.
CANcel	Terminates the dump program and displays status.
DA	See DISPLAY command.
DC	See DISPLAY command.
DEC	Converts a hexadecimal value to decimal.
DF	See DISPLAY command.
DIN	See DISPIND command.
DISPAct	Same as DISPLAY but assumes actual rather than relative addresses.
DISPChar	Same as DISPLAY but only character representation of data is displayed rather than both character and hexadecimal.
DISPFwd	Same as DISPLAY except that the scan/locate pointer is advanced by the length of the display.
DISPInd	Displays an area of storage whose address is determined by a base and a displacement. The base address can be the contents of a general register.
Display	Displays the contents of program storage, general registers or floating point registers.

Figure 6-1 (Part 1 of 2). Summary of Dump Commands

COMMAND	FUNCTION
DUmp	Obtains a display of all general and floating point registers as well as all user program storage.
End	Terminates the dump program.
Eoj	See END command.
Forward	Advances the scan/locate pointer by a specified number of bytes.
HEX	Converts a decimal value to hexadecimal.
Locate	Locates a string of data characters within your object program area.
ORigin	Sets the basis for relative to actual address calculation to a location other than the program load point.
Point	Sets the scan/locate pointer to a specified address.
SAVE	Obtains a hardcopy dump.
SEarch	Locates a string of data characters within your program area.
SHow	Obtains various status displays.
STatus	See SHOW command.
SUB	Subtracts one hexadecimal number from another or subtracts one hexadecimal number from the contents of a general register.
Top	Sets the scan/locate pointer to the first position within the program or defined reference area.

Figure 6-1 (Part 2 of 2). Summary of Dump Commands

ADD Command

The ADD command is used to add two hexadecimal values and to display the sum. It can also be used to add the contents of a general purpose register and a hexadecimal value.

```
ADD            hexval1  ┌ hexval2|0 ┐
               GPRn     └           ┘
```

hexval1	is a one to eight hexadecimal digit value to be added to hexval2.
GPRn	is the designation of a general purpose register where 'n' can be 0 through 9 or A through F.
hexval2	is a one to eight hexadecimal digit value.

The first operand (hexval1) is added to the second operand (hexval2) and the hexadecimal result is displayed on your screen. If the hexadecimal result is an address within your program, its value relative to the start of your program (or to the address specified in an ORIGIN command) is also displayed.

If the first operand specified is a general purpose register (GPRn), the contents are added to the value specified as the second operand.

If the second operand is omitted, a value of zero is used as the second operand.

This command is useful when dealing with base-displacement addressing. For example, in COBOL a field in working storage may be at a displacement of X'04C' from base locater 1. If base locater 1 is general register 6, the command ADD GRP6 04C will return the actual and relative addresses of the desired field (see also the DISPIND command).

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
add 24A 137C
K439I  HEX VALUE IS 000015C6
K404D  ENTER DUMP COMMAND
*ENTER DATA?
add gpr6 4C
K439I  HEX VALUE IS 00B76A8 REL=000640
```

BACKWARD Command

The BACKWARD command causes the scan/locate pointer to be reduced by the specified number of bytes.

```
Backward      [-n|h|-16]
```

-n is a decimal integer indicating the number of bytes by which the scan/locate pointer is to be reduced. If no operand is specified, the pointer is reduced by 16 bytes.

h is a hexadecimal integer indicating the number of bytes by which the scan/locate pointer is to be reduced.

If the number of bytes specified exceeds the number of bytes between the current ORIGIN command point (or load point) and the current scan/locate pointer, the pointer is set to the current origin (usually the load point). That is, the pointer may never be backed up farther than the start of the reference area (ORIGIN command point).

Other dump commands which affect the scan/locate pointer are FORWARD, POINT, TOP, SEARCH, LOCATE, and DISPFWD.

After setting the scan/locate pointer, the new value can be referenced as '*' as in the example below.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
display
080170 000030 000030 4110001095007DF5...etc.
K404D  ENTER DUMP COMMAND
*ENTER DATA?
back -24
K441I  ACTUAL=08158  REL=000018
K404D  ENTER DUMP COMMAND
*ENTER DATA?
disp *
080158 000018 000018 D203850A7C76...etc.
```

CANCEL Command

The CANCEL command is used to terminate the dump program.

```
CANcel        [NOprint]
```

NOprint specifies that all displays are to be bypassed and only the termination messages are to be displayed.

When the CANCEL command is entered, you will receive the following information:

- The termination status information,
- The type of halt condition which caused the termination as well as its location in the program, and
- The contents of the 16 general purpose registers.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
cancel
K401I  LOAD HI-PTN HI-PHS ORIGIN SCAN *** STATUS ***
ACTUAL:   0B7870 0B8C1D 0B7870 0B789E
RLATIV:   000000 0013AD 000000 00002E
K442I  PSW= ... etc.
               ...
GP 0-F    00000050 000B78D8 ...
          A00B2E40 0A16180C ...
K433I  DUMP PROGRAM CANCELED
```

DEC Command

The DEC command is used to obtain the decimal equivalent of a hexadecimal number.

```
DEC           hexval
```

hexval   is a 1 to 8 digit hexadecimal number.

If the hexadecimal number is greater than 7FFFFFFF, it is assumed that the value represents a two's complement value and a negative decimal number is returned.

Examples:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
dec 1ac
K440I  DECIMAL VALUE IS 0000000428
K404D  ENTER DUMP COMMAND
*ENTER DATA?
dec ffffffff
K440I  DECIMAL VALUE IS -0000000001
```

DISPIND Command

The DISPIND command is used to display an area of storage where the exact address to be displayed is not known.

```
DISPInd      hexval1  [hexval2|0 [length|+16] ]
DIN          GPRn
```

hexval1 is a one to six digit hexadecimal value which is interpreted as an actual base address to be added to a displacement address (hexval2) to form a location to be displayed.

GPRn is the general purpose register where 'n' can be 0 through 9 or A through F. The contents of the register is a base address to be added to a displacement address (hexval2) to display a location.

hexval2 is a one to six digit hexadecimal value or an equate to such a value; if omitted zero is assumed. 'hexval2' must be explicitly specified when length is specified (even if 0).

length can be either a hexadecimal value or a decimal value preceded by a plus sign, which indicates the amount of data to be displayed.

What is known is the offset or displacement from another address, which may be in a general purpose register.

If no length is specified, 16 bytes of data will be displayed. If operand1 is specified as a general register, the contents of the register are added to the value specified as operand2. If operand2 is omitted, a value of 0 is assumed.

Notes:

1. *This command is useful with base displacement addressing. For example, in COBOL a field in working storage may be at a displacement of X'04C' from base locator 1. If base locator 1 is general register 6, the command DISPIND GPR6 04C will cause the field to be displayed.*
2. *The /ATTEN command or Attention key*[1] *can be used to terminate display functions and re-display the dump command prompt.*

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
dispind gpr6 4c
K439I  HEX VALUE IS 000B76A8 REL=000640
0B76A8 000640 000640  00374C01982D ...
```

1 The definition of the Display/Attention key is a VSE/ICCF tailoring option. The default is PA3. It may be different for your system.

DISPLAY Command

The DISPLAY command is used to view the contents of program storage, general purpose registers or floating point registers.

```
Display
DISPAct
DA          [address|* [length|+16]]
DISPChar    GPR[n[[GPR]m]]
DC          FPR[n]
DISPFwd
DF
```

address	is the first main storage address to be displayed. If omitted, the address of the scan/locate pointer is used. This operand can be a hexadecimal address or the locate pointer reference '*'.
length	is the hexadecimal or decimal length of the data to be displayed. If a decimal length is intended, it must be preceded by a plus sign. If the second operand is omitted, a length of +16 is assumed.
GPR	requests the display of all general purpose registers.
GPRn	requests the display of general purpose register 'n' only (in this case, the GRPm operand is to be omitted). Or, GRPn is the first of a series of general purpose registers to be displayed, the last register in the series being register 'm' of the GPRm operand. 'n' and 'm' can be 0...9 or A...F.
GPRm	specifies the last of a series of general purpose registers to be displayed, the first in the series being register 'n' of the GRPn operand. The characters 'GRP' may be omitted; 'm' by itself suffices. If the GRPm operand is omitted altogether, only register 'n' of the GRPn operand will be displayed.
FPR	displays all four floating point registers.
FPRn	displays a floating point register, where 'n' can be 0, 2, 4 or 6.

The four commands have similar functions. However note the following differences:

- In the DISPLAY command, address is assumed to be an address within your program area. If a hexadecimal address is specified as operand1, it is assumed that this address is relative to the start of your program (or relative to the address specified in an ORIGIN command). When storage is displayed, both hexadecimal and character representations of storage are displayed. The DISPLAY command does not change the scan/locate pointer.
- The DISPCHAR (display character) command causes only the character representation of storage to be displayed. Instead of 32 hex digits and 16 characters per line, the display is 32 characters per line.
- In the DISPACT (display actual) command, any storage address specified is assumed to be an actual as opposed to a relative address.

- The DISPFWD (display forward) command only differs from the DISPLAY command in that the scan/locate pointer is advanced to the ending display location plus one. This permits repeated DF '*' commands to return successively higher storage locations.

When the display commands are used to print the contents of storage, the display always begins with three 6 digit hexadecimal addresses. The first is the actual storage address of the data, the second is the address relative to the program's load point, and the third is the address relative to the start of your reference area (initially set at the load point and optionally modified by the ORIGIN command).

Following these addresses is the data itself. The data (except for the DISPCHAR command) is displayed in both hexadecimal and character format. If more than 16 bytes are displayed, the hexadecimal display is grouped into full words with each word separated by a space.

When GPRn GPRm is specified, the general registers are displayed in the order indicated. Thus, 'GPRE GPR1' would result in a display of registers 14, 15, 0 and 1. Note the following:

1. The DISPLAY command can be used to obtain the actual equivalent of a relative address, since both actual and relative addresses are displayed on the print line. Conversely, the DISPACT command can be used to obtain the relative address equivalent of an actual address.

2. The /ATTEN command or Attention key[2]
   can be used to terminate display functions and to re-issue the ENTER DUMP command prompt.

[2] The definition of the Display/Attention key is a VSE/ICCF tailoring option. The default is PA3. It may be different for your system.

Examples:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
display 134A +200
K446I  ACTUAL LOAD/REL ORG/REL - STORAGE DISPLAY
0B4730 001340 001340 05F00700 ...  *.0.. ...
0B4740 001350 001350 00085C80 ...  *..*. ...
  . . . (etc.)
K404D  ENTER DUMP COMMAND
*ENTER DATA?
disp
0B3350 000000 000000 05F047F0F03E ... .0.00.
K404D  ENTER DUMP COMMAND
*ENTER DATA?
d gpr
GP 0-F 00000050 00098FFF ...
       900F28A2 000B6400 ...
K404D  ENTER DUMP COMMAND
*ENTER DATA?
d gpr3 5
GP 3-5 0000FF90 000B5DC8 500B6436
K404D  ENTER DUMP COMMAND
*ENTER DATA?
locate 'this'
00B39C 00004C 00004C  E3C8C9E240C9 ... THIS IS...
K404D  ENTER DUMP COMMAND
*ENTER DATA?
df 5C
00B3AC 00005C 00005C  C6D6D9 ... FOR...
K404D  ENTER DUMP COMMAND
*ENTER DATA?
df
00B3BC 00006C 00006C  C5D5C440 ... END ...
K404D  ENTER DUMP COMMAND
*ENTER DATA?
back -16
K441I  ACTUAL=00B3BC REL=00006C
K404D  ENTER DUMP COMMAND
*ENTER DATA?
dc
00B3BC 00006C 00006C  END OF MESSAGE...
```

DUMP Command

The DUMP command is used to display all general and floating point registers as well as all program storage areas.

```
DUmp          [ALL]
```

ALL specifies that the entire interactive partition area is to be displayed.

The format of the dump is the same as that obtained by using the DISPLAY command, except that areas of storage which are all binary zeros are not displayed.

If no operand is specified for the DUMP command, all general and floating point registers as well as the entire program area up to the end of the last phase loaded will be dumped. (If hardcopy is desired see SAVE command.)

Example:

```
K404D  ENTER DUMP COMMAND
dump
```

END Command
EOJ Command

The END or EOJ commands terminate the dump program.

```
End
Eoj
```

Example:

```
K404D   ENTER DUMP COMMAND
*ENTER DATA?
e
K434I   DUMP PROGRAM ENDED
```

FORWARD Command

The FORWARD command advances the scan/locate pointer by the specified number of bytes.

```
Forward        [+n|h|+16]
```

+n is a decimal integer indicating the number of bytes by which the scan/locate pointer is to be advanced. If no operand is specified, the pointer is advanced 16 bytes.

h is a hexadecimal integer indicating the number of bytes by which the scan/locate pointer is to be increased.

If the number of bytes specified would advance the pointer beyond the end of the program, a message is issued and the pointer is not changed.

Other dump commands which affect the scan/locate pointer are: BACKWARD, POINT, TOP, SEARCH, LOCATE and DISPFWD.

After setting the scan/locate pointer, the new value can be referenced by '*' as in the example below.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
locate d207
080EB8 000E48 000E48  D20756AC543A...etc.
K404D  ENTER DUMP COMMAND
*ENTER DATA?
forward +4
K441I  ACTUAL=080EBC REL=000E4C
K404D  ENTER DUMP COMMAND
*ENTER DATA?
disp *
080EBC 000E4C 000E4C 543A41110008...etc.
```

HEX Command

The HEX command is used to obtain the hexadecimal equivalent of a decimal number.

```
HEX         decval
```

decval is a 1 to 8 digit decimal number preceded by a plus or minus sign which is to be converted to hexadecimal notation.

The hexadecimal number is returned as a full word.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
hex +128
K439I  HEX VALUE IS 00000080
K404D  ENTER DUMP COMMAND
*ENTER DATA?
hex -2
K439I  HEX VALUE IS FFFFFFFE
```

LOCATE Command

The LOCATE command is used to find a particular combination of characters in a program.

```
Locate        hexval|'string'
```

hexval	is from 2 to 16 hexadecimal digits 0 through 9 or A through F. An even number of hex digits is required.
'string'	is from 1 to 8 EBCDIC characters bounded by apostrophes. The trailing apostrophe is only required if the character string has one or more embedded or trailing blanks.

The LOCATE dump command alters the scan/locate pointer. The other commands which alter the scan/locate pointer are POINT, FORWARD, BACKWARD, SEARCH, TOP and DISPFWD.

The scan for the desired data always begins at the location of the scan/locate pointer and proceeds to the end of your program. If the desired data is not located, a message is issued and the scan/locate pointer is set to the beginning of the program area (current ORIGIN value).

If you want to begin your search at the beginning of your program rather than at the scan/locate pointer address, issue the TOP command prior to the LOCATE, or use the SEARCH command.

If you are scanning for more than one occurrence of the same string, the pointer must be advanced before each LOCATE, otherwise you will continually find the data at the same location. The FORWARD command (F +1) can be used to do this.

Notes:

1. *The LOCATE command is useful in locating data items within a program when a symbol map is not available. Unique identifiers can be included as constants within or in front of the actual data areas in the source program. The LOCATE can then be used to find the data locations. For example, in a COBOL program you might code the following:*

   ```
   01 WORKAREA
   03 FILLER PIC X(5) VALUE '*CTL*'
   03 CTLFIELD PIC X(6).
   ```

   The LOCATE can be used to find the field named CTLFIELD as in the first example below.

2. *The /ATTEN command (or Attention key[3].) will terminate a LOCATE function that is in progress.*

[3] The definition of the display/attention key is a VSE/ICCF tailoring option. The default is PA3. It may be different for your system.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
top
K404D  ENTER DUMP COMMAND
*ENTER DATA?
locate '*ctl*'
0D348A 000B82 000B82 5CC3E3D35C...   *CTL*...
K404D  ENTER DUMP COMMAND
*ENTER DATA?
forward +32
K441I  ACTUAL=0D34AA  REL=000BA2
K404D  ENTER DUMP COMMAND
*ENTER DATA?
dispfwd * +64
   .
   . (etc.)
   .
```

ORIGIN Command

The ORIGIN command is used to change the start of the reference area that is used as the base in calculating relative to actual addresses.

```
ORigin         address [REL]
               START
               *
               GETvis
```

address is the actual address of the start of the reference area.

REL specifies that the previous address is not an actual address but is relative to the program load point.

START indicates that the reference area (origin) is to be re-established at the load point of the program which is its initial value prior to any ORIGIN commands.

* indicates that the reference area origin will be the scan/locate pointer value.

GETvis indicates that the reference area origin will be the start of the interactive partition GETVIS area.

This command can be used to dump storage in a control section, subprogram or subroutine other than the primary program being tested. The actual address where the subroutine was loaded can be obtained from the link edit or load map and specified as the operand for this command. From this point on, all relative addresses used in commands would be relative to the start of the subroutine.

The command could also be used in referring to a very large record area whose format you know. The origin could be set to the start of this area.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
org 86c80
K420I  ORIGIN ADDRESS=086C80, REL=003410
K404D  ENTER DUMP COMMAND
*ENTER DATA?
disp 1ba
086E3A 0035CA 0001BA  F0F0F3F7F8C2...   00378B...
K404D  ENTER DUMP COMMAND
*ENTER DATA?
org start
K420I  ORIGIN ADDRESS=083870, REL=******
```

POINT Command

The POINT command is used to set the scan/locate pointer to a specified relative location within the program (reference area).

```
Point          [+n|h|+0|GETvis]
```

+n — is a decimal integer indicating that the scan/locate pointer is to be set '+n' bytes into the reference area (ORIGIN command). If no operand is specified, '+0' is assumed.

h — is a hexadecimal value indicating that the scan/locate pointer is to be set 'h' bytes into the reference area. The pointer will be set to the relative address indicated by 'h'.

GETvis — specifies that the scan/locate pointer is to be set to the start of the interactive partition GETVIS area.

Other dump commands which affect the setting of the scan/locate pointer are SEARCH, LOCATE, FORWARD, BACKWARD, DISPFWD, and TOP.

The POINT command is equivalent to a TOP command followed by a FORWARD command. If no operand is specified in the POINT command, zero is assumed, thus making the command equivalent to the TOP command.

If the operand specifies a value which would advance the scan/locate pointer to an address outside the program, a message is issued and the pointer is not changed.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
point 13a
K441I  ACTUAL=0D348A  REL=000882f
K404D  ENTER DUMP COMMAND
*ENTER DATA?
dispchar *
0833AA 00013A 00013A  THE MESSAGE IS H ...
```

SAVE Command

The SAVE command is used to obtain a hardcopy dump of the interactive partition.

```
SAVE          [comment]
```

comment is a character string which can be up to 24 characters long.

This command is useful when you want a hardcopy dump of the interactive partition for problem determination. The comment could, for instance, be a short description of the error as a reminder of why the dump was taken. The dump will first be written to the VSE dump library, and then it can be displayed or printed with Info/Analysis.

Examples:

1. Before the dump is written to disk a message with the dump identification appears on the screen. This is followed by another message with the same identification indicating that the dump is now complete.

   ```
   K404D  ENTER DUMP COMMAND
   *ENTER DATA?
   save invalid addr in pgmabc
   0S30I DUMP STARTED. MEMBER=00000002.  DUMP IN SUBLIB=SYSDUMP.F1
   1I51I DUMP COMPLETE.
   K442I DUMP 00000002 SAVED
   K404D  ENTER DUMP COMMAND
   *ENTER DATA?
   ```

   The identification SYSDUMP.F1.00000002 is the selector to be used for displaying or printing the dump via Info/Analysis.

2. Make sure that the dump library is assigned (LIBDEF DUMP,CATALOG=SYSDUMP.F1,PERM), and that it exists on your system. It should also contain enough free space if you want a hardcopy dump to be taken. In case the dump could not be written to disk for the reasons mentioned above, an appropriate message will be returned.

   ```
   K404D ENTER DUMP COMMAND
   *ENTER DATA?
   save pgm check in calc95
   K436I DUMP LIBRARY FULL OR NOT DEFINED - DUMP NOT SAVED
   K404D Enter DUMP command
   *ENTER DATA?
   ```

SEARCH Command

The SEARCH command is used to find a particular character string within the program. It begins its scan operation at the first location in your program (or the first location in an area specified by the ORIGIN command), and is thus equivalent to a LOCATE command preceded by a TOP command.

```
SEarch        hexval|'string'
```

hexval	is from 2 to 16 hexadecimal digits 0 through 9 or A through F. An even number of hexadecimal digits is required.
'string'	is from 1 to 8 EBCDIC characters bounded by apostrophes. The trailing apostrophe is only required if the character string contains trailing blanks.

The SEARCH command sets the scan/locate pointer to the address where the scan was satisfied. If no match can be found, the scan/locate pointer is set to the first position in the scan area. Other dump commands which set the scan/locate pointer are TOP, POINT, FORWARD, BACKWARD, LOCATE and DISPFWD.

See the description of the LOCATE command for more information and an example of how to use this command.

SHOW Command

STATUS Command

The STATUS command is used to obtain various displays of information within the program.

```
STatus        [INstr [addr]|PSW]
SHow
```

INstr points to an instruction within the program.

addr is the relative hexadecimal address of an instruction within your program.

PSW specifies that the termination program status word is to be displayed.

If no operand is specified in the STATUS command, the actual and relative addresses of all key program control factors are displayed together with information concerning the program termination status.

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
status
K401I     LOAD HI-PTN HT-PHS ORIGIN  SCAN*     ***STATUS***
ACTUAL: 0836E8 0872BB 0839B8 0836E8 0836F0
RLATIV: 000000 003BD3 0002D0 000000 000008
       .
       .
       .
```

The INSTR operand allows you to point to an instruction within your program and to have the dump program decode that instruction into actual and relative data locations, data lengths and the actual contents of the data fields. If no second operand is specified, the termination instruction will be decoded. If an address operand is supplied, it must point to a valid instruction. For example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
st instr
K443I  INSTR=FA2150A950AC
K445I  OPRND2 ACT=083796 REL=0000AE LEN=002 CONTENTS=
083796 0000AE 0000AE    010C0000341C ...
K445I  OPRND1 ACT=083793 REL=0000AB LEN=003 CONTENTS=
083793 0000AB 0000AB    00370C010C00 ...
```

The PSW operand is used to return the termination program status word. The program status word is an internal control area which contains information concerning the termination status of the program being dumped. The rightmost 6 hexadecimal digits of the PSW contain the execution termination address.

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
status psw
PSW=071D0007D008375E
ACTUAL=083758 REL=000070
INSTR=FA2150A950AC
```

SUB Command

The SUB command is used to subtract one hexadecimal value from another hexadecimal value or from the contents of a general purpose register, and to display the result.

```
SUB           hexval1 [hexval2]
              GPRn
```

hexval1	is a 1 to 8 digit hexadecimal number.
hexval2	is a 1 to 8 digit hexadecimal number.
GPRn	is the designation of a general purpose register where 'n' can be 0 through 9 or A through F.

The second operand is subtracted from the first operand and the hexadecimal result is displayed. If the hexadecimal result is an address within your program, its value relative to the start of the program (or your ORIGIN value) is also displayed.

If the first operand is specified as GPRn, the second operand value is subtracted from the contents of the specified general register.

If the second operand is omitted, a value of zero is used as the second operand.

See also the ADD Dump Command.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
sub 234C 82A
K439I  HEX VALUE IS 00001B22
K404D  ENTER DUMP COMMAND
*ENTER DATA?
sub gpr5 800
K439I  HEX VALUE IS 000BA64C REL=0035D4
```

TOP Command

The TOP command sets the scan/locate pointer to the first position in the current reference area.

```
Top
```

If no ORIGIN command has been issued, the current reference area begins at the original load point of the program being tested.

The TOP command can be used before a LOCATE command to ensure that the search begins at the start of the reference area.

Example:

```
K404D  ENTER DUMP COMMAND
*ENTER DATA?
top
K404D  ENTER DUMP COMMAND
*ENTER DATA?
loc 'findit'
K423I  CANNOT LOCATE DATA TOP FORCED
```

Chapter 7. Writing Procedures and Macros

This chapter presents

- The general characteristics of procedures and macros.
- A summary of the IBM-supplied procedures and macros.
- Rules for calling procedures and macros.
- Rules for writing your own procedures and macros.

Procedures and macros are VSE/ICCF library members that contain VSE/ICCF system and editing commands, job entry statements and/or data. The commands, statements and data appear in the VSE/ICCF library member just as they would if they had been entered directly from the terminal. They carry out frequently used utility functions, such as compilation, loading and running programs, storing object decks and sorting library members.

The main advantage of procedures and macros is that they save you work. Once they have been created and stored in your library, they can be run at any time simply by invoking the procedure or macro name. You thus do not have to reenter the statements and commands each time you need a particular function.

The main difference between procedures and macros is that macros are run in the foreground like normal commands and can be invoked in edit mode, whereas procedures are run only in command mode. Procedures require that execution of the procedure processor program be started in an interactive partition, which means that a macro is processed more quickly than a procedure.

Another difference is that a procedure has more control over the flow and execution of the commands than a macro. A macro has only a limited logic capability. Procedures are invoked with the /EXEC command, or simply by specifying the procedure name. Macros issued in edit mode must be prefixed with '@', but this prefix is not necessary in command mode. Both procedures and macros can be written using variables, which allow you to supply them with data.

Macros and procedures can be stored permanently as library members; macros can be built in the stack area. The latter is especially useful while the terminal is in edit mode. Certain macros and procedures are supplied with VSE/ICCF, but you can also write your own procedures and macros.

Most of the IBM-supplied procedures and macros are described in detail in *Chapter 3. System Commands, Procedures and Macros*; you find them there in alphabetical order, together with the system commands. Editor macros are described in *Chapter 4. Editor*. Figure 7-1 on page 7-3 contains a summary of the IBM-supplied procedures and macros.

A discussion of how to write your own procedures and macros appears later in this chapter.

Your installation may have chosen to not install *all* IBM-supplied procedures and macros. To determine if a given macro or procedure is present, list the macro or procedure. For example, to check for the STORE macro, enter /LIST STORE.

Summary of IBM-Supplied Procedures and Macros

PROCEDURE OR MACRO	FUNCTION
ASSEMBLE	Causes a library member to be processed by the VSE assembler.
COBOL	Causes a library member to be processed by the DOS/VS COBOL compiler.
@COPY	Copies lines from one part of a file to another
COPYFILE	Creates a copy of a library member under a specified name in your primary library.
COPYMEM	Creates a copy of a library member under a specified name in a specified library.
CPYLIB	Copies a library member from one library to another.
$DA	Displays the status of VSE partitions.
ED	Invokes the full screen editor directly.
EDPRT	Places the terminal in full screen edit mode for editing of the print area.
EDPUN	Places the terminal in full screen edit mode for editing of the punch (stack) area.
FORTRAN	Causes a library member to be processed by the VS FORTRAN compiler.
@FSEDPF	Sets PF keys for use in full screen edit mode.
GETL	Retrieves VSE/POWER list queue output and places it in a library member, or in the print area.
GETP	Retrieves VSE/POWER punch queue output and places it in a library member, or in the punch area.
GETR	Retrieves a job enqueued in the VSE/POWER reader queue and stores it as a member of the VSE/ICCF library file.
HC	Switches an IBM 3270 terminal to hardcopy mode, executes a specified command and returns to normal display mode.
HELP	Displays how-to-use information on VSE/ICCF commands.
LIBRC	Catalogs a VSE/ICCF library member into a VSE sublibrary.
LIBRL	Lists a member of a VSE sublibrary and, optionally, stores it into a VSE/ICCF library member.

Figure 7-1 (Part 1 of 2). Summary of IBM-Supplied Procedures and Macros

PROCEDURE OR MACRO	FUNCTION
LIBRP	Punches a member of a VSE sublibrary and, optionally, stores it into a VSE/ICCF library member.
LOAD	Causes an object program to be loaded into storage and run.
@MOVE	Moves lines from one part of a file to another.
MVLIB	Moves a library member from one library to another.
PLI	Causes a library member to be processed by the DOS/VS PL/I optimizing compiler.
PRINT	Routes the contents of a library member, or the input area, to a hardcopy printer associated with a 3270 terminal for printing.
RELIST	Routes the contents of the print area to the printer.
RPGIAUTO	Invokes AUTO REPORT and compiles with RPG II.
RPGII	Causes a library member to be processed by the DOS/VS RPG II compiler.
RPGIXLTR	Invokes the RPG II DL/I translator.
RSEF	Calls the RSEF program (the RPG II Source Entry Facility)
SCRATCH	Removes DISP=KEEP files from the dynamic space area after they are no longer required.
SDSERV	Displays the directory (sorted) of a primary, connected or the common library.
SORT	Sorts a library member and places the output as directed.
$SPACE	Causes the DTSSPACE program to be run in an interactive partition.
STORE	Saves the contents of the punch area as a library member.
SUBMIT	Submits VSE/ICCF or VSE job streams to be run in a a batch VSE partition.
VSBASIC	Is used to compile and punch or compile and run a VS BASIC program, or load and run an object deck.
VSBRESEQ	Is used to resequence an VS BASIC source program.

Figure 7-1 (Part 2 of 2). Summary of IBM-Supplied Procedures and Macros

Spacing and Delimiters

The spacing of operands and the use of delimiters between operands is the same as for normal system commands with the following exceptions:

1. Operands in macros cannot use parentheses as delimiters.

2. Operands in procedures can be bounded with apostrophes if the operand itself contains delimiter characters.

Passwords

Many of the procedures are not set up to handle member passwords. If the members you are dealing with are password protected, you must enter the commands and job entry statements individually or by building your own alternate procedures.

Operand Limitations

Procedures may have open-ended parameter lists ending with one or more options. Note that the options specified must not cause the total number of operands to exceed 9. If the number of operands would otherwise exceed 9, two or more of the options can be specified as a single operand by grouping them together within apostrophes. Thus, 'NODECK LIST XREF' is considered to be one operand.

Implied Execute Function

In order to run procedures (as opposed to macros) by simply entering the procedure name, the implied execute (IMPEX) feature must be turned on; see the /SET IMPEX system command on page 3-115. If it is turned off, the /EXEC CLIST command must be entered. For example, ASSEMBLE PROGA would have to be entered as /EXEC ASSEMBLE CLIST PROGA if the implied execute feature had been set off.

Input Area

All procedures and some macros will cause the contents of the input area to be lost. Thus, if the input area is to be retained, it should first be saved before invoking the procedure or macro.

Rerunning a Procedure

When a procedure is invoked, the VSE/ICCF job stream to be run is built in the input area. Thus, as long as the options do not change, the procedure can be run whenever it is needed simply by entering /RUN (or $). Also, if the job stream thus created is to be used again, the input area can be saved (/SAVE command) and rerun by entering /EXEC followed by the name of the library member where the job stream is saved. Rerunning a procedure in this way is more efficient than reentering the procedure name and operands since the overhead of the procedure processor execution need not be repeated.

Editing by Using Macros and Procedures

You can automate editing tasks by using a procedure or a macro. The following is an overview of the available methods together with their specific advantages.

1. Edit macros in full screen edit mode.

   Advantage: A macro executes fast. An edit macro may contain full screen editor commands.

   Limitation: Return messages from edit commands cannot be checked. Variable user input (for substitution in edit commands or to control conditional skipping of statements) is only possible through parameter passing.

2. Macros calling (in command mode) the context editor.

   Advantage: A macro executes fast. Checking of return messages from edit commands is possible.

   Limitation: Variable user input (for substitution in edit commands or to control conditional skipping of statements) is only possible through parameter passing.

3. Procedures calling (in command mode) the context editor.

   Advantage: System variables and internal variables can be used. Via a dialog with the user, the procedure can request variable data for control or substitution.

   Limitation: Execution in an interactive partition is slower. Or, an interactive partition may not be available for some time.

Writing Your Own Procedures

Writing your own procedures means creating a VSE/ICCF file that can be run whenever you need an often-used function. The file contains VSE/ICCF system and editing commands, job entry statements, procedure processor orders and/or data. The procedure orders must not have sequence numbers in columns 73-80.

A Simple Procedure

For example, to write a procedure which would save punch output from a previous job into the library under the name JCPUNCH, you might write the following procedure:

```
/INPUT
/INSERT $$PUNCH
/END
/PURGE JCPUNCH
/SAVE JCPUNCH
```

If you entered this procedure as input (edit mode) and saved these statements in the library under the name SAVPUN, then you could run the procedure (SAVPUN) consisting of the above statements simply by entering:

```
/EXEC SAVPUN CLIST
```

or:

```
SAVPUN
```

At this point the procedure processor (DTSPROCS) would be called and the commands in the procedure would be executed one by one until the last line was processed just as if the commands had been entered from the terminal.

Variable Parameters in Procedures

The facility described above is useful; however, it would be difficult to build truly generalized procedures without supplying some type of data to the procedure. In the above example, the punch output may only be saved in the library under the name JCPUNCH.

It would be more useful to be able to specify the name of a library member in which to save the punch output at the time the procedure is invoked. This can be done by specifying a variable parameter name (in this case, &&PARAM1) in place of JCPUNCH so that the procedure appears:

```
/INPUT
/INSERT $$PUNCH
/END
/PURGE &&PARAM1
/SAVE &&PARAM1
```

Now when this procedure is run a parameter must be specified to replace the variable parameter (&&PARAM1) when it is encountered in the procedure. Thus the procedure is invoked as follows:

```
/EXEC SAVPUN CLIST OBJMOD
```

or

```
SAVPUN OBJMOD
```

Then, when the procedure is invoked and the /PURGE and /SAVE commands are encountered, the word OBJMOD will replace the variable parameter (&&PARAM1) and the punch output will be saved in the library under the name OBJMOD.

Up to 18 variables (&&PARAM1 through &&PARAM9 and &&PARAMA through &PARAMI) may be used in this way to supply variable information to a procedure. (Even more than 18 variables can be accessed via the &&SHIFT order.)

Logic in Procedures

It may not always be enough to simply replace variables in a procedure. Sometimes you may want to interrogate the variables themselves, so that you can alter the flow of commands as they are processed within the procedure.

Suppose in the preceding example you wanted to save the punch output in the library under the name specified as above but, if no name was specified, you wanted to have the punch output saved under the name JCPUNCH. In the above procedure, if the member name operand is simply omitted, the variable parameter (&&PARAM1) will be replaced by blanks and the /PURGE and /SAVE commands will be invalid. Thus, the procedure is modified so that it appears:

```
/INPUT
/INSERT $$PUNCH
/END
&&IF &&PARAM1 EQ ' ' &&GOTO AAAA
/PURGE &&PARAM1
/SAVE &&PARAM1
&&EXIT
&&LABEL AAAA
/PURGE JCPUNCH
/SAVE  JCPUNCH
```

In this example, the &&IF order is used to determine if the parameter has been omitted. (We are, in fact, checking to see if the parameter is blank.) If so, a branch (&&GOTO order) is taken to a point in the procedure called AAAA(&&LABEL). At this point the default commands for saving the output as JCPUNCH will be executed.

The Procedure Processor

The following paragraphs discuss concepts and facilities of the procedure processor. They are important to both the procedure writer and to the procedure user.

Procedure Defined

A procedure is a library member containing system commands, edit commands, job entry statements, procedure processor orders and/or data that performs specific functions. Procedures are processed in the interactive partition by a utility program called DTSPROCS. The /LOAD for DTSPROCS can be the first statement in the procedure, in which case the procedure can be invoked by entering a simple /EXEC command (for example /EXEC PRTPCH). If the /LOAD DTSPROCS is not the first statement in the procedure, a CLIST operand of the /EXEC command must be used to invoke the procedure (for example /EXEC PROC CLIST).

Variable Procedure Defined

A variable procedure is similar to a procedure except that certain command operands can be represented by variable symbols in the form &&PARAMn, where n is a digit from 1 to 9 or a letter from A to I. The variable procedure can also contain logical orders which are described later in this section. An order is a statement which is interpreted by the procedure processor to perform a logical function or alteration of the flow of the procedure. The variable procedure must be invoked with the CLIST form of the /EXEC. The actual parameters to be substituted for the variable symbols are coded as operands on the /EXEC command (for example /EXEC LOAD CLIST FORTPROG DATA FORTDATA).

Implied Execute Function

Unless the implied execute function has been set off (see the /SET IMPEX system command on page 3-115), there is a simpler way of entering procedures. In the implied execute form, the /EXEC and the CLIST keyword can be omitted so that the above example can be entered as LOAD FORTPROG DATA FORTDATA.

DTSPROCS Program

The DTSPROCS program is the procedure processor. It reads system and edit commands as if they were data and passes the commands to the proper processing module just as if the commands had been entered from the terminal. The DTSPROCS utility also monitors the execution of /RUN and /EXEC requests which themselves are part of the procedure.

The ability to issue /RUN or /EXEC requests within a procedure is a standard facility of the DTSPROCS program. However, when more than one execution is requested in a procedure, two interactive partitions are required to process the request: one for execution of DTSPROCS and the second for the executions requested by the /RUN or /EXEC commands. The CLIST form of the /EXEC command cannot be used in a procedure. That is, a procedure cannot invoke another procedure.

The DTSPROCS program can also scan procedures for variable parameters (&&PARAMn) and for procedural orders (for example &&IF). When a variable parameter is encountered in the procedure, the actual parameter from the invoking /EXEC command is substituted. If no parameters are supplied on the /EXEC command, the variable parameters are set to blanks.

DTSPROCS can also be used to transfer 80-character records from the library into the punch area. If the UPSI-0 switch is set on, the printed data will also be transferred to the punch area if a /DISPLAY, /LIST or /LIBRARY command is encountered within the procedure. This facility is useful if, for example, you want to process the output of the /LIBRARY command. If the UPSI-1 switch is set on, the punch data will not be displayed. If the UPSI-2 switch is set on, the normal display output from the command processor will not be displayed.

DTSPROCS is to some extent a typewriter terminal user of its own. So it has its own control blocks and its own auxiliary areas (input, print, punch, stack, log area). Only the contents of the print area are transferred to your print area automatically. For the contents of the other areas you have to take care in the procedure. If you for example started logging in a procedure, you would have to list the log area in the same procedure, since at procedure end the log area would be lost.

Notes to the DTSPROCS Program:

1. The input to DTSPROCS consists of 80-character records. It is best to avoid sequence numbers or flagging information in columns 72-80, since they may be interpreted as data. Characters in column 72 are not interpreted as continuation characters.

2. DTSPROCS always assumes an IBM 2741 terminal. Commands that require a 3270 terminal, such as /HARDCPY, cannot be issued from a procedure. This explains why a procedure cannot get control in list mode. All listing operations proceed without pause to their end. You thus cannot give system commands valid in list mode within procedures: for example /COMPRESS, /CONTINU, /SKIP.

3. An input area created by a procedure (for example by the /INPUT command), is lost at the end of the procedure unless it is saved by a command issued from within the procedure. An exception to this rule is when a /RUN or /EXEC is followed by a /PEND control statement. Then the input area remains available to the job as if it had been built via terminal input.

4. If you password protect a procedure you rule out implied execution. The password would be taken as the first parameter.

5. PF keys cannot be set or shown from a procedure.

The CLIST Operand of the /EXEC Command

The /EXEC command in the form:

```
/EXEC listname CLIST param1 ...
```

or:

```
listname  param1 ...
```

places the following statements in the input area:

```
/LOAD DTSPROCS
/PARM param1 ...
/INCLUDE listname
```

and then the job is run. Thus, the CLIST option of the /EXEC command is nothing more than an automatic way of invoking DTSPROCS. The /PARM control statement is produced only if parameters are specified following the CLIST.

The /PARM Control Statement

The /PARM control statement is used to specify the actual values to be substituted for any variable parameters encountered within the 80-character records by DTSPROCS. The format of the statement is:

```
/PARM param1 param2 ...
```

Normally the /PARM statement is generated automatically by the /EXEC or an implied /EXEC with one or more specified parameters, as above; however, there are certain instances where you may want to invoke DTSPROCS explicitly and thus code your own /PARM statement.

The values specified in the /PARM statement can be separated by blanks or commas. Each value can consist of from 1 to 30 characters. If a parameter contains delimiter characters such as commas or blanks, it may be enclosed in quotes. The first value (param1 above) will be substituted for the variable parameter &&PARAM1, the second value for &&PARAM2, etc. If you want a parameter to be omitted, indicate the omission by specifying a single ampersand (&). The total length of all values and delimiters may not exceed 72 characters and the number of values should not exceed 20. The /PARM statement is generated automatically by a CLIST /EXEC or implied /EXEC command with one or more parameters specified.

The /PEND Control Statement

If a /PEND statement is supplied prior to the last or only /RUN or /EXEC within the procedure, it will prevent two interactive partitions from being required for the execution of the final (or only) step. When the PEND condition is on, all records following a /RUN or /EXEC are flushed (PEND conditions: either a preceding /PEND statement was present or the MULTEX option is off).

Ampersand (&) Coded Job Entry Statements

Normally when the VSE/ICCF background job stream reader encounters a job entry statement such as /LOAD, /DATA or /INCLUDE, the statement is processed as it is encountered. That is, the /LOAD will mark the end of one step and the start of the next, or an /INCLUDE referenced member will be logically included at that point. However, it is often desirable to be able to build job streams within procedures. Thus, we would not want an /INCLUDE or /LOAD processed at the point it is encountered as data but, rather, later as a job entry statement in the job stream being built within the procedure. To provide this capability, job entry statements to be read as data by DTSPROCS can be specified starting in column 2 with an ampersand in column 1. Thus, /LOAD becomes &/LOAD. In fact, any statement encountered by the procedure processor with &/ in columns 1 and 2 will be shifted left 1 column. Note that /* or /& will terminate a procedure; therefore &/* or &/& should be used instead.

Procedure Processor Variables

The variables recognized by the procedure processor fall into three major categories: 1) variable parameters (&&PARAMn), 2) internal variables (&&VARBLn and &&COUNTn) and 3) special variables like &&DEVTYP. Any variable's value can be tested by specifying the variable as operand 1 of an &&IF order. In addition, all variables except &&CURLN can be specified in any command, data or order input to the procedure processor. The value assigned to the variable will be substituted for the variable name before the statement is processed by the procedure processor.

Note: Variables are not substituted in the job entry statements /DATA, /INCLUDE, and /LOAD.

Because all variable names are 8 characters in length and predefined, a concatenation character is not needed. Variables can be concatenated to other variables or character strings simply by specifying the variables adjacent to each other with no intervening spaces.

The following paragraphs describe the variables which you can reference in your procedures:

Variable parameters:

```
&&PARAMn (where n is 1 through 9 and A through I)
```

The &&PARAMn variables are set in a /PARM statement from values which were specified when the procedure was invoked. For example, if /EXEC PROCA CLIST 25 67 were entered, the value of &&PARAM1 would be 25 and the value of &&PARAM2 would be 67. The value specified should not be more than 30 characters in length. All the &&PARAMn variables can be reset to new values while the procedure is running via the &&READ order.

Internal Variables:

`&&VARBLn (where n is 0 - 9 and A - I)`

There are 19 internal alphameric variables for temporary storage of intermediate values. These variables can be set using the &&SET order.

`&&COUNTn (where n is 0 - 9 and A - I)`

There are 19 internal numeric variables for temporary storage of intermediate values. These variables can be set, incremented or decremented using the &&SET order.

Special Variables:

`&&CURDAT`

This variable contains the date consisting of six digits. The format depends on the format of the // DATE JCL statement.

`&&CURTIM`

This variable contains the time of day consisting of six digits in the 'HHMMSS' (hour, minute, second) format.

`&&CURLN(mm,nn)`

This variable can only be used as operand 1 of an &&IF order. Its use implies that the editor current line is to be tested for some character string whose starting position within the line (01 to 80) is indicated by mm and whose length (01 to 32) is indicated by nn. Also if mm is equal to zero (00), every column within the editor current line is tested for the string. This variable allows you to build editor procedures which apply logic to the data being edited.

`&&DEVTYP`

This variable is set to 70 if the procedure is being run from a 3270, to 41 if from a 2741 (or 3767 in 2741 mode), to 40 for any other typewriter terminal or to 00 for a sequential device acting as a terminal. Thus, the procedure can interrogate the terminal device type if the procedure logic depends on the terminal type.

`&&EXRC`

This variable contains the job execution return code as set by the program that was called (as opposed to &&RETCOD which is the VSE/ICCF return code). Compilers, utilities and some user programs return a job execution return code.

The return code is a decimal number between 0 and 4095. If the program was called from a procedure or from a macro, this return code is passed to the macro and procedure processors as a 4-byte character string.

You as the terminal user might get, for example, the following completion message

***** JOB TERMINATED - ICCF RC 00,EXEC RC 0004, NORMAL EOJ

where

ICCF RC	is the VSE/ICCF return code as documented in the publication *VSE/ICCF Messages*
EXEC RC	is the job execution return code as set and documented by the program that was called.

```
&&LINENO
```

This variable always contains a numeric value equal to the line number of the current line being processed within the procedure.

```
&&PARMCT
```

This variable always contains a numeric value from 0 to 20 indicating the number of operands specified when the procedure was invoked.

```
&&RDDATA
```

This variable contains the first word from the line which was entered in response to a &&READ order with no operands.

```
&&RETCOD
```

This variable contains the VSE/ICCF return code. The contents of this variable is one of the following: if in edit mode it contains the first word of the response message issued by the execution of the preceding command. For normal system commands, &&RETCOD will contain *READY, or the first word of the last * response message before the *READY or *END PRINT message. The return code after an execution (/EXEC or /RUN) requested within a procedure will be *EOJ-XX, where XX is the VSE/ICCF return code. (00 is normal end-of-job). In any case, the &&RETCOD value does not exceed 8 characters. The &&RETCOD variable can be tested to determine if a command was executed successfully.

Example:

```
/switch 12
*SWITCHED
*READY
```

If you want to test for successful switching, check the character string '*SWITCHE' in &&RETCOD.

```
&&TERMFT
```

This variable contains the string 'DBCS' or 'KATAKANA', if the procedure has been invoked from an IBM 5550 with 3270 Emulation. It can only be used as operand 1 of an &&IF order.

```
&&TERMID
```

This variable contains the terminal identification of the terminal on which the procedure was executed.

```
&&USERID
```

This variable contains the userid of the user who invoked the procedure.

Summary of Procedure Processor Orders

When you write procedures that perform logical operations, you use special commands called orders, which are recognized only by the procedure processor. For example, with the &&IF order you can interrogate operands that you have specified and then alter the flow of control in the procedure based on the outcome of the test.

Here is a summary of the procedure processor orders described on the next pages:

ORDER	FUNCTION
*	Is used to place comments within the procedure.
&&EXIT	Causes an exit from the executing procedure.
&&GOTO	Causes the transfer of control to a specific location in the procedure.
&&IF	Is used to conditionally execute statements within the procedure.
&&LABEL	Is used to identify branch points within a procedure.
&&MAXLOOP	Is used to alter the maximum loop value that determines when the procedure will be canceled.
&&NOP	Causes all variable parameter substitution and procedure order checking to be bypassed.
&&OPTIONS	Sets certain options that control the execution of the procedure processor.
&&PUNCH	Causes any operand data in the order to be placed in the punch area.
&&READ	Reads a single line of data from the terminal.
&&SET	Is used to alter the contents of the internal variables &&VARBLn and &&COUNTn.
&&SHIFT	Shifts parameters to the left to make variable parameter processing easier.
&&TYPE	Enables you to display data.

Figure 7-2. Summary of Procedure Processor Orders

* Order

The * order is used for comments within the procedure.

```
 *              comment
```

When an * order is encountered in command mode it will not be treated as a VSE/ICCF command but as a comment to be ignored, although it will not be ignored for input, edit and execution modes.

&&EXIT Order

The &&EXIT order causes an exit from the executing procedure.

```
&&EXIT
```

The &&EXIT order can be used in situations where you want to terminate processing in a procedure before actually reading through to the end of the procedure. For example, you might use the &&EXIT order within a loop when the condition for exiting from the loop is recognized.

&&GOTO Order

The &&GOTO order causes the transfer of control to a specific location in the procedure.

```
&&GOTO        [±|-] label|nn
```

+	specifies that the &&LABEL point which is branched to follows the &&GOTO which references it. + is default.
–	specifies that the &&LABEL point which is branched to precedes the &&GOTO which references it.
label	specifies the name on a &&LABEL order elsewhere in the procedure.
nn	specifies a decimal integer from 1 to 255 which indicates a positive or negative skip amount.

A &&GOTO order must reference a statement in the same member. It may not branch over an /INCLUDE statement.

For example, &&GOTO -1 would cause the preceding statement to be repeated and &&GOTO +2 would cause the following statement to be bypassed.

A &&GOTO order cannot be the last statement in a procedure. This restriction can be overcome by following a &&GOTO at the end of a procedure with a dummy &&LABEL order.

&&IF Order

The &&IF order is used to conditionally execute statements within the procedure.

```
  &&IF        variable  operator  data  then-clause
```

variable	can be any of the variables described under *Procedure Processor Variables* earlier in this section.
operator	can be any of the following: EQ, NE, LT, GT, LE, or GE; or the symbols `=, ¬=, <, >, <=, >=.` This parameter is used to describe the type of condition to be tested.
data	can be a variable (&&PARAMn, &&COUNTn, &&VARBLn, &&RDDATA, &&RETCOD, &&LINENO, or &&PARMCT), a numeric literal (for example 123 or -14), or an alphameric literal. If blanks are in the string the literal must be within quotes (for example READY or 'END OF LINE').
then-clause	can be a system or editing command, a line of data, another order or any other line or command which would be processible in the procedure at the point where the &&IF order occurs.

The &&IF order tells the procedure processor to determine whether the expressed condition is true or false. If the expressed condition is false, no further processing of the &&IF order takes place and the next sequential line in the procedure is processed.

If the condition is true, the then-clause (which is any operand information beyond the third operand) is, in effect, shifted into column 1 and reevaluated as a command, data line or another order. Compound conditions can be built on the same statement by coding multiple &&IF orders on the same line.

The comparison between operands 1 and 3 is always alphameric unless operand 1 is specified as a variable having a numeric characteristic such as &&COUNTn, &&PARMCT or &&LINENO. In these cases, the comparison is algebraic. In alphameric comparisons the shorter operand is right padded with blanks to the length of the longer operand before the compare is made.

&&LABEL Order

The &&LABEL order is used to identify branch points within a procedure.

```
&&LABEL     label
```

label	must begin with an alphabetic character and must be from 1 to 8 characters in length. Any 'branch to point' specified as a label on a &&GOTO order must be identified by a &&LABEL order.

&&MAXLOOP Order

The &&MAXLOOP order is used to alter the maximum permissible loop value before the procedure is automatically canceled.

```
&&MAXLOOP    [nn]
```

nn is a decimal number from 1 to 4096.

The &&MAXLOOP order can only appear before the first &&LABEL order within a procedure. If &&MAXLOOP is not set, a default value of 150 is used for the maximum loop value.

Each time a &&GOTO order is encountered within a procedure, a counter is incremented by 1. If this counter reaches the maximum loop value the procedure will be terminated.

&&NOP Order

The &&NOP order causes all variable parameter substitution and procedure order checking to be bypassed for the entire statement.

```
&&NOP      data
```

data specifies that the data beginning in column 7 will be shifted into column 1 and the resulting statement will be passed to the command processor.

If you want to pass data to the command processor but this data began, for example, with &&IF or some other characters which the procedure processor would recognize as an order, the data should be coded beginning in column 7 of a &&NOP order.

&&OPTIONS Order

The &&OPTIONS order sets options that control the execution of the procedure processor.

```
&&OPTIONS    abcdefgh
             00001000
```

The characters 'a' through 'g' refer to options that can be set on by specifying (1) or off by specifying (0). An 'x' leaves the option unchanged.

The options have the following meanings:

a If this option is set on, any display output from list type commands such as /LIST, /DISPLAY, /LIB or PRINT (editor) will be punched (placed in the punch area) rather than displayed. This option is also set by the '10000000' UPSI option.

b If punching (option 'a') is set on, the punch data will normally be displayed as well as punched. However, if this option is set off, the data will not be displayed. This option is also set by the '01000000' UPSI option.

c If this option is set on, the normal command processor display output will not be displayed. Only critical procedure error messages and &&TYPE order requests will be displayed; all other display output will be bypassed. This option is also set by the '00100000' UPSI option.

d When this option is set on, no shifting of data will occur. For example if a 3-character parameter '***' replaces an 8-character variable name '&&PARAM1' in the string '123&&PARAM1456' the resulting string is '123***bbbbb456'. When this option is off, the resulting string is '123***456'.

e This option (which is called the MULTEX option) will initially be set to '1' or on. When it is off, any /RUN or /EXEC within the procedure will signal the end of the procedure. The procedure processor will be terminated and the execution request will be processed in the same interactive partition. When this option is on, the procedure processor will stay in the interactive partition (to process any possible commands following the /RUN or /EXEC) and the execution request will be scheduled in a second interactive partition. However, if a prior /PEND control statement had been encountered, the result will be as if this option had been set off.

f When this option is off, user control options such as tabs, control characters, etc., will be set and maintained just as if the commands had been entered from the terminal. If this option is on at the time the procedure terminates, these same control factors will not have a lasting effect. They would be set and utilized in the procedure but their effect would not be maintained for normal operation.

g When this option is off, the procedure processor will halt immediately prior to any /RUN or /EXEC request encountered in the interactive partition and any output in the print area will be forced out to the terminal. When this option is on, the halting and forcing of the print area does not occur.

h Not used.

&&PUNCH Order

The &&PUNCH order causes any operand data on the order to be placed in the punch area.

```
&&PUNCH      data
```

data specifies that the data beginning in column 9 of the &&PUNCH order will be placed in column 1 of an 80-character record.

The operand data can contain variable parameters in which case normal variable parameter substitution will occur.

&&READ Order

The &&READ order causes a single line of data to be read from your terminal when the order is encountered in an executing procedure.

```
&&READ        ┌          ┐
              │&&VARBLn  │
              │&&COUNTn  │
              │&&PARAMS  │
              └          ┘
```

&&VARBLn specifies that the data read will be placed into the internal variable. The data entered must not exceed eight characters.

&&COUNTn specifies that the data read will be placed into the internal variable. The data entered must be a decimal value and can be preceded by a plus or minus sign.

&&PARAMS specifies that a new set of parameters will be read in to replace &&PARAM1 through &&PARAM9 and &&PARAMA through &&PARAMI. The old parameters are first cleared, then the new ones are read in. You indicate the omission of a parameter by specifying a single ampersand (&).

If no operand data is present, the entire line which is read from the terminal is passed to the command processors just as if the command or data line had been in the procedure at the point where the &&READ occurred.

Following the &&READ, the first word read can be tested using the &&RDDATA variable.

Note: If a valid command is entered in response to 'ENTER DATA', the command is executed by the foreground command processor. The procedure processor does not get control. When command execution is finished, 'ENTER DATA' will be displayed again.

&&SET Order

The &&SET order is used to alter the contents of the internal variables &&VARBLn and &&COUNTn. In addition, the &&COUNTn variable can be incremented (or decremented) from its current value:

```
&&SET        &&VARBLn [&&SUBSTR i j]  character string
                                      &&PARAMn
                                      variable

             &&COUNTn  ┌ ± ┐mm
                       │*+ │
                       │*- │
                       └   ┘
                       &&PARAMn
                       variable
```

n	value from 0 - 9 or A - I for &&VARBLn or &&COUNTn; a value from 1 - 9 or A - I for &&PARAMn.
&&VARBLn	When setting one of the &&VARBLn internal variables, you can specify a character string up to 8 characters in length. If the character string contains delimiter characters, it must be bounded by apostrophes. Or you can set the internal variable to a parameter (&&PARAMn). If &&PARAMn consists of more than 8 characters, only the first 8 characters will be used to set the variable. &&VARBLn can also be set to another internal variable (&&VARBLn or &&COUNTn) or any other special variable except for &&CURLN (e.g., &&RETCOD or &&LINENO).
&&SUBSTR	The &&SUBSTR order can immediately follow the first operand to set an internal variable (&&VARBLn) to a sub-portion (substring) of the character string (or parameter, or any variable except &&CURLN) specified as the last operand. The 'i' operand must be specified as the starting location within the character string (1 to 32) and the 'j' operand must be specified as the number of characters to be set (1 to 8). Thus, if &&PARAM1 contains ABCDEF, the order &&SET &&VARBL3 &&SUBSTR 2 3 &&PARAM1 will cause &&VARBL3 to be set to BCD.
&&COUNTn	When setting a numeric internal variable (&&COUNTn), you can specify a plus or minus decimal integer ('±mm operand, + is default) to cause the variable to be set to a given value. To increment or decrement a variable, you can specify asterisk plus or minus the decimal increment ('*+mm' operand). In addition, the &&COUNTn variables can be set to parameters (&&PARAMn) or other variables (&&VARBLn, &&COUNTn, &&LINENO, etc.) as long as these variables contain numeric values.

Note: Only character strings shorter than eight characters can be enclosed in apostrophes. Longer strings may not contain apostrophes, blanks, commas, asterisks or brackets.

&&SHIFT Order

The &&SHIFT order causes parameters to be shifted one position to the left as a means of making variable parameter processing easier.

```
&&SHIFT      n
```

n is a decimal number from 1 to 9 indicating the parameter at which point the shift is to begin.

Given the parameter list A B C D E F, a procedure could reference these internally as &&PARAM1, &&PARAM2, ... &&PARAM6. If the order &&SHIFT 3 was specified, the parameter list would then be A B D E F. Every parameter beyond &&PARAM3 was shifted to the left one logical location.

This is very useful in procedure processing where a variable number of keywords can be coded as parameters and coded in any order. If the first non-fixed parameter was &&PARAM3, you could check &&PARAM3 for each of the variable keywords. Then when one was found, you could process it or save the fact that it was entered in a variable (&&VARBLn) and then say &&SHIFT 3 and repeat your test loop. The parameters that had been &&PARAM4, 5, 6, etc., will now be &&PARAM3, 4, 5, etc., so &&PARAM3 can be tested again and again until there are no more parameters (&&PARAM3 equals a blank).

Up to 19 parameters can be made available in this way.

The &&SHIFT order reduces the value of the &&PARMCT variable by one.

&&TYPE Order

The &&TYPE order enables you to display data.

```
&&TYPE      data
```

data	can consist of any (alphanumeric or double-byte) data characters including variables which will be replaced by their current values before the display takes place.

Even if the no print option is in effect (see &&OPTIONS order) a &&TYPE order will cause data to be displayed.

Examples of Procedures

Example 1: Assemble a Program

The following procedure will assemble a source module and, optionally, save an object module in a library member rather than in the punch area. You can also type in additional operands which are treated as assembler or VSE/ICCF options.

```
*--------------------------------------------------------------
* PROCEDURE TO ASSEMBLE A PROGRAM
*
* 1ST PARM IS THE 'NAME-OF-MEMBER' CONTAINING THE SOURCE
*
* 2ND PARM IS OPTIONAL KEYWORD 'OBJ' FOLLOWED BY 'NAME-
*  OF-MEMBER' INDICATING THE VSE/ICCF MEMBER TO CONTAIN
*  THE OBJECT DECK -- IF NOT ON THE FILE IT WILL BE
*  CREATED
*
* ADDITIONAL PARMS ARE TREATED AS OPTIONS AND ARE PLACED
*  ON THE /OPTION STATEMENT
*--------------------------------------------------------------

&&IF &&PARAM2 NE OBJ &&GOTO INLIB
/LIST 1 1 &&PARAM3  (see if object member in library)
&&IF &&RETCOD EQ *READY &&GOTO INLIB (if normal return, it
/INP                                  is in library)
DUMMY
/SAVE &&PARAM3   (build dummy member for object module)
&&LABEL INLIB
/INP
&/LOAD ASSEMBLY
&&IF &&PARAM2 EQ OBJ &&GOTO +OBJ1 (test if object deck to
                                   library member)
/OPTION &&PARAM2 &&PARAM3 &&PARAM4 &&PARAM5 &&PARAM6 &&PARAM7
&&LABEL OBJ1
&&IF &&PARAM2 NE OBJ &&GOTO +NOOBJ1
/OPTION &&PARAM4 &&PARAM5 &&PARAM6 &&PARAM7
/FILE TYPE=ICCF,UNIT=SYSPCH,NAME=&&PARAM3 (define member
&&LABEL NOOBJ1                                 for object deck)
/FILE NAME=IJSYS01,SPACE=40,VOL=0
/FILE NAME=IJSYS02,SPACE=40,VOL=1
/FILE NAME=IJSYS03,SPACE=20,VOL=2
&/INCLUDE &&PARAM1
/END
/PEND
/RUN
```

If the above procedure is invoked as follows:
```
     assemble beep obj myobj list xref
```
the following lines will be generated and passed to the command processors:

```
/LIST 1 1 MYOBJ
*NOT IN LIBRARY (asterisk lines are responses from command
*END PRINT       processors)
*READY
/INP
DUMMY
/SAVE MYOBJ
*SAVED
*READY
/INP
/LOAD ASSEMBLY
/OPTION  LIST XREF
/FILE TYPE=ICCF,UNIT=SYSPCH,NAME=MYOBJ
/FILE NAME=IJSYS01,SPACE=40,VOL=0
/FILE NAME=IJSYS02,SPACE=40,VOL=1
/FILE NAME=IJSYS03,SPACE=20,VOL=2
/INCLUDE BEEP
/END
*READY
/RUN
```

Example 2: Execute a Program

This LOAD procedure can be used to load and run object decks from the punch area or from a library member.

```
*-------------------------------------------------------------
* PROCEDURE TO EXECUTE OBJECT PROGRAMS
*
* 1ST PARM IS NAME OF MEMBER CONTAINING OBJECT DECK -OR-
  '*' INDICATING DECK IS IN PUNCH AREA
*
* OPTIONAL KEYWORD 'JES' FOLLOWED BY 'NAME-OF-MEMBER'
*  JOB ENTRY STATEMENTS ARE IN 'NAME-OF-MEMBER' -OR-
*  '*' INDICATES THE STATEMENTS ARE TO COME FROM THE
*  TERMINAL
*
* OPTIONAL KEYWORD 'DATA' FOLLOWED BY 'NAME-OF-MEMBER' -OR-
*  '*' DATA IS IN 'NAME-OF-MEMBER' '*' INDICATES THE
*  'INCON' OPTION IS TO BE USED
*-------------------------------------------------------------
&&SET &&VARBL0 &&PARAM1
&&IF &&PARAM2 EQ JES &&SET &&VARBL1 &&PARAM3
&&IF &&PARAM4 EQ JES &&SET &&VARBL1 &&PARAM5
&&IF &&PARAM2 EQ DATA &&SET &&VARBL2 &&PARAM3
&&IF &&PARAM4 EQ DATA &&SET &&VARBL2 &&PARAM5
&/INP
&/LOAD LINKNGO
&&IF &&VARBL1 EQ '*' /PAUSE -- TYPE JOB ENTRY STMTS HERE
&&IF &&VARBL1 NE ' ' &&IF &&VARBL1 NE '*' /INCLUDE &&VARBL1
&&IF &&VARBL0 NE ' ' /UPSI 1
&&IF &&VARBL0 NE ' ' /INCLUDE &&VARBL0
&&IF &&VARBL2 EQ '*' &&SET &&VARBL3 INCON
&&IF &&VARBL2 NE ' ' /DATA &&VARBL3
&&IF &&VARBL2 NE ' ' &&IF &&VARBL2 NE '*' /INCLUDE &&VARBL2
/END
/PEND
/RUN
```

If the above procedure is invoked as follows:

```
     load myobj jes myjes data mydata
```

the following lines will be generated and passed to the command processors:

```
/INP
/LOAD LOADER
/INCLUDE MYJES
/UPSI 1
/INCLUDE MYOBJ
/DATA
/INCLUDE MYDATA
/END
/RUN
```

Example 3: Edit a File

The following procedure causes a file (&&PARAM1) to be scanned. Each line in the file which contains the character string R1 and also the character string R7 will be located by the procedure and displayed on the terminal and the terminal will be prompted for an editing command. At this point you can enter a command to cause the data to be changed.

```
/ED &&PARAM1
BRIEF
NEXT
&&LABEL AAAA
&&OPTIONS XX1    (set printing off)
BRIEF
NEXT
&&IF &&RETCOD EQ INVALID &&EXIT
&&IF &&CURLN(00,2) EQ R1 &&IF &&CURLN(0,2) EQ R7 &&GOTO CCCC
&&GOTO -AAAA
&&LABEL CCCC
&&OPTIONS XX0    (set printing on)
VERIFY
&&TYPE ENTER EDIT COMMAND
&&READ     (read edit command)
&&IF &&RDDATA EQ QUIT &&EXIT (test command entered)
&&IF &&RDDATA EQ Q &&EXIT
&&IF &&RETCOD EQ *READY &&EXIT
BRIEF
&&GOTO -AAAA
&&LABEL DUMMY
```

Writing Your Own Macros

What is a Macro?

A macro is a library member containing VSE/ICCF commands, procedure invocations and macro orders. Macros make it easy and efficient for you to invoke often-used, complex functions. You simply issue the macro name like a command -- remembering (if you are in edit mode) to prefix it with the '@' sign. In command mode this prefix is no longer necessary to invoke a macro, but it is still valid. The invocation of a macro is only allowed on the last statement of the current macro. A macro cannot have a sequence number or flag field in columns 73-80.

Variable Parameters

Macros can contain variable parameters &&PARAM1 through &&PARAM9 as operands of VSE/ICCF commands. These parameters are replaced with the actual operands that you later specify when the macro is called. Each parameter value can be eight bytes long, plus the number of spaces that you place between parameters when you write the macro. Shorter parameters are padded with blanks. The operands in macros must not contain blanks or commas because these would be treated as delimiters. Variable parameters may no longer be available after commands that cause execution or list mode to be entered. If you want to omit an operand in a macro call, you have to specify && as place holder, unless this is the last operand. The last operand can be omitted.

Temporary Macros

Macros can be built in the punch area or editor stack area and invoked by specifying @$$PUNCH or @$$STACK. This is useful in the editor, where you may want to repeat a series of commands. First put the commands (and/or data) into the stack using the STACK command. Then, when you want to execute them, enter @$$STACK. Remember that the stack area is part of the punch area.

Note: A macro built in the stack area must not contain the @MACRO statement as the first statement in the macro.

Chaining Macros

A macro cannot be called from another macro unless the call appears as the last statement of the calling macro. When a macro call is encountered within a macro, it is treated as if it were the last statement in the containing macro.

Abnormal Termination of a Macro

When an invalid command condition occurs (for example an ADD command without a parameter), the macro execution terminates immediately.

Macro Orders

The first statement within a macro must be '@MACRO'. It must start in column 1. All macro orders must start in column 1. The '@NOPRINT' macro order can be inserted in a macro to prevent the normal output from the commands in the macro from reaching the terminal. This helps eliminate unwanted intermediate terminal output. The '@PRINT' macro order reverses this effect. A macro order must not be the last statement in a macro, otherwise it would be treated as a macro call.

@BACK and @FOR Macro Orders

The @BACK and @FOR macro orders are used to cause a backward or forward branch within the macro to re-execute certain statements. These orders can be used with the @LIMIT or @IF orders.

```
@BACK     nn
@FOR      nn
```

nn is the number of statements to branch forward or backward. It can be a value between 1 and 99999999. You can also specify a variable parameter for nn.

@IF Macro Order

The @IF macro order is used to test whether the preceding command in your macro has been executed successfully. If the @IF condition is false, the next sequential statement in your macro is executed. If the @IF condition is true, you can either branch forward or backwards a specified number of statements.

```
@IF     ┌        ┐                          ┐
        │&&EXRC  │                          │
        │&&PARMCT│  EQ|NE    data           │
        │&&PARAMn│                          │  @BACK nn
        │&&RETCOD│                          │  @FOR nn
        └        ┘                          │  @EXIT
                                            │  command
        &&CASOUT    EQ|NE UPPER|CTL         │

        &&TERMFT    EQ|NE DBCS|KATAKANA     │
                                            ┘
```

&&EXRC contains the return code of the last interactive partition execution.

&&PARMCT allows you to check for the existence of parameters at macro invocation time. You can compare only with 0: if no parameters were specified, &&PARMCT EQ 0 holds, else &&PARMCT NE 0 is true.

&&PARAMn allows you to check the contents of the passed parameters &&PARAM1...&&PARAM9.

&&RETCOD contains the first word of the response message issued by the execution of the preceding command (for a more detailed description of &&RETCOD see "Procedure Processor Variables" on page 7-11).

data can be a variable (&&PARAMn) or a literal of up to 8 characters. If blanks are in the string, the literal must be within quotes.

&&CASOUT allows you to check the output case settings 'UPPER' and 'CTL'.

&&TERMFT contains the string 'DBCS' or 'KATAKANA', if the macro was invoked from an IBM 5550 with 3270 Emulation.

@EXIT Macro Order

The @EXIT macro order causes an exit from a macro before the physical end of the macro is reached. @EXIT at the physical end of a macro is ignored.

`@EXIT`	`command`

command	can be any VSE/ICCF command, but not a macro order.

Examples:

1. Exit the macro and edit member MYFILE:

```
@EXIT /ED MYFILE
```

2. Exit both the macro and the editor if the comparison is equal:

```
@IF &&RETCOD EQ DATA @EXIT QUIT
```

@LIMIT Macro Order

The @LIMIT macro order is used to set the maximum number of statements that will be processed in the macro before it is terminated.

@LIMIT	nn

nn is the maximum number of statements that will be processed. nn can be a value from 1 to 32767. The default is 150. nn can also be a variable parameter.

@MACRO Macro Order

The @MACRO macro order must be the first statement within a macro.

@MACRO	

@NOPRINT Macro Order

The @NOPRINT macro order is used to eliminate the normal output from the commands in the macro from reaching the terminal. This helps avoid unwanted intermediate terminal output.

@NOPRINT	

Note: @NOPRINT is reset by a conversational read or a full screen read/write operation. A program or a procedure started from within a macro can issue such a terminal I/O request.

@PRINT Macro Order

The @PRINT macro order is used to route normal output from the commands in the macro to the terminal.

@PRINT	

Examples of Macros

1. The following macro puts your terminal into edit mode and sets up all your options and defaults. You invoke this macro (called ED) by specifying ED xxxx, where xxxx is the name of the member to be edited.

```
@MACRO
@NOPR
/ED &&PARAM1
NEXT
SET LOG ON
SET HEX=<
SET TAB=;
SET END=:
SET SCR 4
TABSET 10 16 36 68
VERIFY LONG 80 15 9
LINEMODE 73,6
PROMPT 4
INDEX
SET PF1ED OC68 CLR//
SET PF2ED UP 10
SET PF10ED STACK OPEN
SET PF11ED STACK CLOSE
SET PF4ED @$$STACK
SET PF12ED P $$STACK
SET PF8ED GETFILE $$STACK
SET PF9ED P $$LOG
SET PF3ED PF 10,80
PRINT
SET SCR CLEAR
ECHO *READY TO EDIT &&PARAM1
```

2. The following example illustrates the use of a macro within a macro that will convert a library member from lower to upper case. This might be useful if you had accidentally entered data in lower case mode when you had intended it to be upper case. This macro builds a temporary macro and finally executes it.

```
@MACRO
@NOPR
/INP
/INSERT &&PARAM1
/END
/SET CASE UPPER
/ED
I @MACRO
I @NOPR
I /INP
B
I /END
I /REPL &&PARAM1
I @PRI
I /ECHO *&&PARAM1 IS NOW UPPER CASE
SAVE TMP$$$$$
@TMP$$$$$
```

Chapter 8. Utility Programs

The following VSE/ICCF utility programs are available to you. You can use some of them directly; others are available via the submit-to batch facility.

DTSAUDIT	displays the changes that were made to a given library member following the last system checkpoint.
DTSBATCH	allows you to execute a limited number of foreground commands in a VSE partition. DTSBATCH can only be invoked via the submit-to-batch facility.
DTSCOPY	copies one sequential file to another.
DTSDUMMY	performs no processing function; it goes immediately to end-of-job. Is often useful in procedures where you might want to sometimes run a program and sometimes not, depending on variable parameters.
DTSSORT	performs a fast, in-core sort on data from the input area or punch area.
LINKNGO	links programs and subprograms together in storage to form an executable program, which is then run.
OBJECT	transfers 80-character records from the job stream to the punch area. Its principal purpose is to transfer object programs or LINKNGO control statements to the punch area for later loading and execution.

DTSAUDIT Utility

The DTSAUDIT utility displays added, deleted and changed records in library members. You would use it to monitor additions, deletions and changes to data within your library for auditability and reliability purposes. The changes displayed will be only those made to the member since the last file checkpoint.

The DTSAUDIT utility program allows you to carry out auditing and tracing. For example you can:

- Scan a given member, or a group of members for changes.
- List only the changes to a member, or the entire member along with its alterations (additions, deletions, changes), including where they occur within the member.
- Select one or more scan options for given members. These options are:
  - A physical scan option, which indicates all physical additions and deletions. This scan also indicates if a member is entirely new. All physical scanning is based on a checkpoint concept. A physical scan can be made on any library member.
  - A logical scan option, which indicates all logical additions, changes and replacements to the member based on the VSE/ICCF editor flagging option. Any record flagged by the editor as a change, replacement or addition is indicated in the change report. This type of scan can be made on any library member but only has meaning for those members which have been changed with the editor change flagging option set on.
  - A sequence number scan option, which scans for sequence number alterations to indicate additions or deletions. This type of scan would only be appropriate for members with imbedded sequence numbers.

Program Concepts

Here is a more detailed description of the three most important of the above options: the physical scan option, the logical scan option and the sequence number scan option. These options can be used singly or in combination on a given member.

Physical Scan Option

The physical scan option displays all physical additions and deletions to the data being scanned since the installation's last checkpoint. The checkpoint concept is important only with the physical scan option.

If an entire member was added following the last checkpoint, the entire member is indicated as new. The physical scan option does not apply to compressed library members whereas the logical and sequence number options described below do apply to compressed members.

Logical Scan Option

The logical scan option displays all changes made to the file based on the editor flagging option. If editor flagging is used within VSE/ICCF to edit certain or all library members, each altered record is flagged as to the date and type of change and the user making the change. The logical scan option of the DTSAUDIT utility causes a member to be scanned for the editor flag. When a flagged record is found, it appears in the report. The report indicates the date and type of change (addition, change, replacement) and the userid of the user who made the change. To use the logical scan option, you should:

1. Set the editor flag option on for all members which are to be edited with this option. This option can be set on using the /PROTECT command with the flag option.

2. Run the DTSAUDIT program with the logical scan option to display all records flagged by the editor.

The Sequence Number Scan Option

The sequence number scan option displays any deletions or additions to a member as indicated by alterations in the normal sequence number progression within the data being scanned. This option is meaningful only for those library members which contain imbedded sequence numbers within the individual records themselves. The sequence number scan is based on an increment value associated with the last resequencing of the member.

The only prerequisite for using sequence number checking is that the data being scanned must contain imbedded sequence numbers and that these numbers were generated using a fixed increment value at the time of the last resequence.

Commands

The DTSAUDIT utility recognizes several different commands. The primary command is the PRINT command. The PRINT command requests an actual DTSAUDIT scan of the library file.

All commands must begin in column 1 of the input control record. Commands must be wholly contained within a single 80-character record. Commands can be abbreviated down to the minimum form as described below. All command operands must be separated from each other by commas or spaces. Multiple commas or spaces are treated as single commas or spaces. All operands can be abbreviated down to the point where they become non-unique among the set of all operands.

The actual commands available are:

- PRINT - The PRINT command requests a specific type of scan. This command controls whether physical, logical or sequence number options (or a combination of 2 or 3 options) will be performed on the requested data.

- OPTION - The OPTION command allows the user to set several of the DTSAUDIT program options on a global basis. That is, once an option is set via the OPTION command, it remains in effect for all later PRINT commands until reset by another OPTION statement. Thus, when an option is set this way, it need not be re-specified on each later PRINT command.

- CARD - The CARD command has no operands. It can be used to set the control statement input device from the console to the card reader.

- END - The END command is used to terminate command input. It would primarily be used when entering commands from the console since the /* or end of file would normally terminate commands entered via SYSIPT.

PRINT Command

```
   Print        [scntyp] [memtyp] [libs] [seqscan] [options]
```

scntyp Is the physical/logical scan type to be performed. The options are:

> LOGICAL - performs a locical scan
> PHYSICAL - performs a physical scan
> BOTH - performs both a logical and a physical scan. BOTH is the default if this operand is not specified.
> NEITHER - performs neither of the above scans

memtyp Indicates which member within the libraries specified in the 'libs' operand will be scanned. The only possible operand when running DTSAUDIT in an interactive partition is 'MEMBER xxxxxxxx PASS yyyy' which must be specified.

The 'MEMBER' operand specifies that only the member whose name is specified (xxxxxxxx) will be scanned. If running in an interactive partition and you are not the VSE/ICCF administrator, the password portion of this operand must also be specified if accessing in a password protected member.

libs May be specified as 'ALL' or 'LIBRARY nnnn'. 'ALL' is the default which indicates that the member specified in the 'memtyp' operand will be scanned in all libraries within the VSE/ICCF library file. If LIBRARY nnnn is specified, only library 'nnnn' will be scanned.

seqscan Indicates that a sequence number scan is wanted. This option can be used by itself or with either the physical/logical or both options. It is specified as 'SEQUENCE n1 n2 n3' where the parameters represent the following values: 'n1' is the starting column number in the record where the sequence number occurs; 'n2' is the number of columns in the sequence number field, and 'n3' is the increment used during the last resequence of the member.

options May be specified as one or several of the following operands:

> SORTED indicates that library directories are to be internally sorted by member name before processing.
>
> EJECT indicates that the printer should be skipped to head of form before printing the output report for each member.
>
> FULLPRINT indicates that all lines within the members being scanned should be printed rather than the default, which is printing only the added, changed or deleted lines.
>
> RESET resets the editor flagged records in any member being scanned using the logical scan option. Thus, the next time the DTSAUDIT program is run, the previously flagged records will no longer be indicated as changed records. RESET causes the editor flag area within a flagged record to be overlayed with four asterisks and the month and day of the RESET. This option is only effective in a VSE batch partition and when VSE/ICCF is not running.

OPTION Command

```
Option        [option1] [option2] ...
```

option1 option2...	Are options to be set for all PRINT commands. Thus, equivalent options need not be specified on each PRINT command. All OPTION command settings are reset each time a new OPTION command is read. The following options can be specified:
SORTED	indicates that all library directories processed by the DTSAUDIT utility are to be sorted alphabetically by member name prior to processing.
EJECT	indicates that a skip to the head of the form is to be performed prior to printing the change report for each member processed by the DTSAUDIT Utility.
NOEJECT	indicates that a skip to head of form is not to be performed for each new PRINT command. Normally, each new PRINT command read from the command input device advances the printer to the head of the form. This option causes this skip to head of form for each PRINT command to be bypassed.
FULLPRINT	causes both the changes and the entire member to be printed. Changed lines will be flagged to distinguish them from the unchanged lines.
RESET	indicates that after a logical scan, editor flagged records will have the editor flag area reset or overlayed within the records so the records do not appear as changed on later DTSAUDIT reports.

Invoking DTSAUDIT:

The basic job stream for executing DTSAUDIT in an interactive partition would be:

```
/INPUT
/LOAD DTSAUDIT
      .
      .  (one or more commands)
      .
/ENDRUN
```

To type in commands interactively, begin execution by entering:

```
$DTSAUDIT CONSOLE
```

Then when the conversational read is requested, enter the appropriate PRINT command.

Examples:

1. Scan all libraries for all occurrences of a member named XFER. Perform a physical, logical and sequence number scan on these members. Print all records, not just flagged records.

   ```
   PRINT MEM XFER SEQ 1 6 100 FULL
   ```

2. Print listing of the member called MYTEST from library 5. Do not perform any of the scanning options.

   ```
   PRINT NEITHER FULLPRINT MEM MYTEST LIB 5
   ```

3. Perform a logical scan on the member called VSMCOB in library 4, and print all records within the member.

   ```
   PRINT LOGICAL FULL MEMBER VSMCOB LIB 4
   ```

4. Determine in which library or libraries the member named ADRSLST occurs.

   ```
   PRINT NEITHER MEMBER ADRSLST
   ```

5. Perform miscellaneous scans on various library members.

   ```
   PRINT LIB 1 MEM ASMPRGA SEQ 73 6 100
   PRINT LIB 1 MEM ASMPRGB SEQ 73 6 100
   PRINT LIB 1 MEM ASMPRGC SEQ 73 6 100
   PRINT LIB 4 MEM COBPRGA SEQ 1 6 100
   PRINT LIB 4 MEM COBPRGB SEQ 1 6 100
   PRINT LIB 6 MEM MYPROC PHYSICAL
   ```

DTSBATCH Utility

The DTSBATCH utility cannot be invoked as part of an interactive partition job stream. It can only be used with submit-to-batch -- that is, it can only be run via the SUBMIT procedure, and not with a /RUN or /EXEC command.

The DTSBATCH utility allows you to execute foreground system and edit commands (except /EXEC, /ENDRUN, /MSG, /RUN, /SEND, VSE/POWER interface commands like /LISTP and /CTLP, or the full screen editor commands) in an off-line submit-to-batch way.

The principal use of the utility is to provide you with a means of printing members from the library on the printer, or of punching members into 80 character records.

The other features of the DTSBATCH utility are not of particular importance to you, since you can perform these same functions by entering the actual commands. However, it is possible to set up lists of commands (except /RUN, /EXEC or VSE/POWER related commands) which can be processed offline in batch mode.

Control Statements

The control statements for the DTSBATCH utility are 80 character records of any foreground command you may want to be executed against the VSE/ICCF file in the batch execution mode.

Bits 3 and 4 of the UPSI byte are used to control punching of library members. If /UPSI 0001 is specified, any time a /LIST, /DISPLAY or /LIB command is encountered the output is directed to the card punch as well as to the printer. When /UPSI 00011 is specified the normally listed output is directed to the card punch but the printing is suppressed.

Notes:

1. *Because in normal input mode, system commands cannot be entered as data, editor input mode must be used for entering the commands. For example, if you are in system input mode and you wanted to enter a command (/LIST member) as data, the /LIST command would be interpreted and the listing would be produced immediately. However, in the input sub-mode of the editor, system commands can be entered and will be accepted as normal data.*

2. *When setting up job streams using DTSBATCH, the /LOGON and password must be entered to allow you to have batch access to the system. Because the logon password must be coded in the data, it may be wise to password protect the job stream, especially if you share a library with others. This protects the security of your logon password.*

Examples:

1. Punch card decks from the members named FORTPROG and FORTDATA in your library.

```
/INPUT
/CANCEL                  (ensure that input area clear)
ED
INPUT                    (enter editor input mode)
/LOAD DTSBATCH     *
/UPSI 00011        *     (punch decks, suppress print)
/LOGON USRJ        *     (logon with your userid)
PASWDJ             *     (your logon password)
/LIST FORTPROG     *
/LIST FORTDATA     *
/LOGOFF            *
                         (null line to return to edit mode)
SAVE BATCHEX             (save commands in the library)
```

   The lines flagged with an asterisk (*) are saved as a library member and form the basis for the batch execution.

2. Produce full directory listing and list the members COBSAMP and BASCSAMP with line numbers.

```
 (enter input sub-mode of the editor as above)
/LOAD DTSBATCH
/LOGON JACL
JACJUN
/LIB FUL,ALL
/DISPLAY COBSAMP
/DISPLAY BASCSAMP
/LOGOFF
```

DTSCOPY Utility

The DTSCOPY utility program copies a disk file from one file area to another (input may also be a tape file). The files may be on the same disk device type or on different device types.

Any file created using the standard VSE sequential disk access method can be copied. Direct access files without keys can also be copied if the file area contains an EOF record at the end of the usable area or if you supply a record count on the control statement defined below.

Any blocksize up to 8000 bytes may be copied.

A typical use for this program is transferring permanent data sets into the temporary file areas (IJSYS00,1,2,3,4). This may be desirable when you are testing an application and do not want to endanger the permanent file. Copying may also be useful when processing in programming languages such as BASIC or FORTRAN which do not offer complete flexibility in naming files.

Control Statement

The DTSCOPY utility requires that one control statement be read. Its format is:

```
/PARM infile [indevice] outfile [outdevice] [STOP=nnnn]
```

The parameters should be separated by blanks or commas. The parameters must be specified in the order indicated.

infile	is the name of the input file; that is, the file to be copied. The name can be from one to seven characters in length. If IJSYSIN is specified as the infile parameter, the statements immediately following the /PARM statement in the input job stream will be written to the output file (blocked 10x80). If IJSYSTP is specified, the unlabeled tape file assigned to SYS004 will be written to the output disk file.
indevice	is optional. It is the device type on which the file resides. The possible operands are 2314, 3330, 3340, 3350, 3375, 3380 or FBA. If this parameter is omitted, FBA will be used.
outfile	is the name of the file area into which the input file is to be copied. The name can be from one to seven characters in length. The CISIZE specification of an 'outfile' residing on an FBA device should be 4K or greater, or omitted entirely. If omitted, 4K is assumed. Therefore, a file may require more space than before if it originally had a smaller CISIZE.
outdevice	is optional. It is the device type on which the output file area resides. It can be specified as 2314, 3330, 3340, 3350, 3375, 3380 or FBA. If it is omitted, the input device type will be used.
STOP=nnnn	is an optional parameter. It can be used to specify the maximum number of data blocks to be transferred. 'nnnn' when specified must be a four digit decimal number. This parameter can be used to terminate the copy operation when there is no EOF record at the end of the file. It may also be useful if you want to transfer only a portion of a file for test purposes.

Examples:

1. Transfer the contents of a file named NAMEFIL into the temporary file area IJSYS04. The default device type for the installation is used and all records are transferred. It is assumed that work files IJSYS01,2,3 and 4 are pre-allocated and require no /FILE statements.

```
/INPUT
/LOAD DTSCOPY
/PARM NAMEFIL IJSYS04
/ENDRUN
```

2. Copy the temporary file area IJSYS00 which is on a 3330 device to the permanent file area named INVMAST which is on 3340. It is assumed that IJSYS00 is a pre-allocated work file requiring no /FILE statement.

```
/INPUT
/LOAD DTSCOPY
/FILE NAME=INVMAST,SER=INVPAK,ID='INVENTORY.MASTER'
/PARM IJSYS00 3330 INVMAST 3340
/ENDRUN
```

3. Copy the first 20 blocks of data from a file named PAYMAST to a file named PAYTEST which will be allocated at the same time from dynamic space.

```
/INPUT
/LOAD DTSCOPY
/FILE NAM=PAYMAST,SER=PAYPAK,ID='PAYROLL.MASTER'
/FILE NAM=PAYTEST,SPACE=6,DISP=KEEP
/PARM PAYMAST PAYTEST STOP=0020
/ENDRUN
```

DTSDUMMY Utility

When run, this utility goes immediately to end-of-job. It may be useful if you want to set /OPTION statement options or /UPSI statement switches in a job step prior to the step which tests the condition.

The DTSDUMMY utility can also be useful in procedures that contain execution dependencies. Here, depending on /EXEC command parameters, you may want a step to be run, and sometimes not. A /LOAD for a variable parameter can be set up. The default for the variable parameter can be defined as DTSDUMMY. If the parameter is supplied on the /EXEC command, the program indicated is run. Otherwise, DTSDUMMY is run.

Another use of DTSDUMMY is to force (/FORCE) terminal print output at a certain point in a job stream, for example between a compile and an execution where otherwise the correct placement of the /FORCE would be difficult.

Examples:

1. Use DTSDUMMY to set options and UPSI switches:

```
/LOAD DTSDUMMY
/UPSI 101
/OPTION SAVEPUNCH,TIME=500
/LOAD . . . (next step)
```

2. Use DTSDUMMY to force print output between a compilation and an execution:

```
/LOAD VFORTRAN
(FORTRAN Source)
/LOAD DTSDUMMY
/FORCE
/DATA
(input data)
```

DTSSORT Utility

The DTSSORT program is an in-core, single pass sort program which can be invoked to sequence records from your input area or punch area. Records from a member can be sorted if they are placed in the input area (with the /INSERT command or the /INCLUDE statement). The sorting sequence is established through the use of a SORT control statement which is read from the input area. It has the following format:

```
    SORT input sequence output
```

SORT - this keyword must begin in column 1. It indicates that this is the sort control statement.

input - the source of the input records must be specified. This operand must follow the SORT keyword by at least one blank. INPUT specifies that the records for the sort are to be read from your job stream; that is, the 80 character records (or an /INCLUDE for the records) to be sorted immediately follow the SORT control statement. PUNCH specifies that the contents of the punch area are to be used for input.

sequence - this operand must follow the input operand by at least one blank. There can be up to 4 subfields for sequencing ranging from major to minor. There are no spaces permitted between subfield specifications. Each subfield specification begins with an A (ascending) or D (descending) followed by the two digit starting column number and the two digit length of the subfield. The fields can overlap in any manner but may not extend beyond the 80th column.

output - the output of the sort is normally to the punch area. It may also be routed to the terminal as display data by specifying PRINT following the sort sequence operand. Or, by supplying an appropriate /FILE statement, output can be an existing VSE/ICCF library member; see Example 5 below. The 'output' keyword must be preceded by at least one blank.

Default Sort:

If the SORT control statement is not read as the first record, then a default control statement is used. The default causes records from the input area to be sorted on columns 1-15 in ascending sequence and the sorted output to be placed in the punch area.

Examples:

1. Sort a portion of a member - output to punch:

```
/INPUT
/LOAD DTSSORT
SORT INPUT A1005
/INSERT MYFILE 2 48
/ENDRUN
```

2. Sort an entire member - output to terminal:

```
/INPUT
/LOAD DTSSORT
SORT INPUT A1005D1503 PRINT
/INCLUDE NAMEFILE
/ENDRUN
```

3. Sort data in punch area - output to punch:

```
/INPUT
/LOAD DTSSORT
/OPTION RESET
SORT PUNCH D0105A1005A4310
/ENDRUN
```

4. Default Sort - input from input area - output to punch, ascending sequence columns 1-15:

```
/INPUT
/LOAD DTSSORT
(data or /INCLUDE referencing the data)
/ENDRUN
```

5. Sort an entire member - output overlays original member in the library:

```
/INPUT
/LOAD DTSSORT
/FILE NAME=NAMEFILE,UNIT=SYSPCH,TYPE=ICCF
SORT INPUT A1005
/INCLUDE NAMEFILE
/ENDRUN
```

LINKNGO Utility

This program is used to bring the object code from a compiler into an executable form and then to transfer control to the entry point for execution. It is implicitly invoked whenever a /RUN or /EXEC request with OPTION DECK or GO is made for a compiler or the assembler. It can be specifically invoked through the control statement /LOAD LINKNGO.

Input to LINKNGO is normally from the punch file, but this can be changed to the input area through the use of the /UPSI 1 statement.

LINKNGO will automatically find any subroutines or system support modules that may be required for program execution. This action is normally transparent to you and should not concern you unless the message UNRESOLVED EXTRN appears as part of the output in the load information (see print output from LINKNGO). Valid input to LINKNGO consists of optional control statements and object modules. All LINKNGO control statements use the standard VSE link-edit control card format.

Control Statements

These are the valid control statements and their formats **(all must be preceded by at least one blank column):**

ACTION - This control statement sets the mode of processing for this run of LINKNGO. Only one ACTION statement is permitted per run and it must precede all other control statements in the input stream. Individual actions must be separated by commas if more than one is used.

```
 ACTION      [CLEAR][,MAP|,NOMAP][,NOAUTO]
```

CLEAR - Causes storage to be cleared to binary zeros up to the end of the program area. Nothing beyond the end of the last control section or the end of common (if present) is cleared. This is to allow uninitialized areas to have binary zeros as their initial value. It is generally better to program the initial values than to use this option.

MAP - Produces a listing of the location of each control section of the program along with its name and all entry points within the control section.

NOMAP - Suppresses the MAP option and is the default option.

NOAUTO - Suppress the automatic lookup in the VSE library for any unresolved external names. This option would be used to ensure that only object modules specifically requested were included in the program.

Other action codes that are valid to VSE will be ignored, since they pertain to situations which are not relevant in an interactive system.

PHASE - This statement gives a name to your program, allows you to specify page boundary alignment or any offset from the start of the interactive partition, and allows the suppression of the automatic lookup in the VSE library. Any other options which are valid under VSE are ignored. This statement must be processed before any object modules are processed.

```
PHASE    name,origin,actions
```

name - 1 to 8 alphameric characters. This will be used as the name of your program and is required if the PHASE statement is submitted.

origin - The program always starts at the beginning of the user's storage area plus 56 character positions unless an additional offset factor is present. Valid origins are '*' and 'S', and one or the other must be present. The offset factor immediately follows the origin and has the format +x'nnnnnn'. The offset value must be six hexadecimal digits enclosed in quotes. This value will be added to the normal program start value and is the new starting point of the program.

actions - These are additional actions which will pertain to the whole operation of LINKNGO.

NOAUTO - same as on ACTION statement. Suppresses the automatic lookup for any unresolved external names. This option would be used to ensure that only object modules specifically requested were included in the program.

PBDY - Depending on the environment in which VSE/ICCF is running, LNKNGO sets the start address of the program to a multiple of 2K or 4K.

INCLUDE - This control statement causes inclusion of the named module from a VSE library. This inclusion takes place as soon as the last statement of the input stream has been read.

```
INCLUDE    name
```

name - Must be the 1-8 alphameric character name of a module in a VSE library. Submodular naming is not permitted.

ENTRY - This control statement, if present, denotes the end of the input stream. It is used to specify an external name which should be given control when the load function is complete.

```
ENTRY    name
```

REPLACE - This statement is used **within** an object module to replace data which has already been assembled or compiled. This statement allows you to change the value of constants and instructions within an object module. The statement must be located after the data it is to replace or it will have no effect. The best position for this statement is before the END statement of an object module. There is no limit to the number of REPLACE statements allowed in an object module because they are processed and applied as read.

```
*REPbbAAAAAAbIIIdata
```

***REPbb** - These must be the first six characters of the REPLACE statement (bb is 2 blanks). The VSE format (first position X'02') is also acceptable but difficult to enter on most terminals.

AAAAAA - This is a six byte hexadecimal address of the start of the data to be replaced. This address is taken from the program listing. Any necessary relocation is done by the loader program.

III - Is the identification number of the control section containing the data to be replaced. It is always hexadecimal and, except for a multiple control section program, is always 001. In the case of the multi-CSECT program, the number can be read from the external symbol dictionary listing on the compiler output.

data - This field immediately follows the III (no intervening blanks). It is always in groups of 2 bytes (four hexadecimal characters) separated by commas. A maximum of 24 bytes can be replaced on a single REPLACE statement.

/UPSI - The /UPSI 1 job entry statement causes LINKNGO to read its input from the job stream rather than from the punch area.

Note: *The LINKNGO program uses the installation default unit (usually SYS008 but it may be some other) for reading object decks from the punch area (unless /UPSI 1 is specified). Therefore, if your program also reads from the punch area using a different SYS number via an /ASSGN, the object deck must be read from SYSIPT using /UPSI 1.*

Examples:

1. Normal compile and run (implicit use of LINKNGO):

```
/INPUT
/LOAD FCOBOL                  (/OPTION DECK assumed)
(source program)
/DATA
(input data)
/ENDRUN
```

2. Combining a module from a VSE library (SUBRTN) with the output of an assembly:

```
/INPUT
/LOAD OBJECT (writes following 3 statements to punch area)
      ACTION MAP,CLEAR,NOAUTO
      PHASE MYTEST,*
      INCLUDE SUBRTN
/LOAD ASSEMBLY
 (program)
/DATA
 (data)
/ENDRUN
```

3. Running entirely from a VSE library:

```
/INPUT   (implicit use of LINKNGO)
/LOAD OBJECT
      INCLUDE PROGA
/ENDRUN
or
/INPUT   (explicit use of LINKNGO)
/LOAD LINKNGO
/UPSI 1
      INCLUDE PROGA
/ENDRUN
```

4. Executing an object program saved in the VSE/ICCF library file:

```
/INPUT
/LOAD OBJECT
/INCLUDE name
/ENDRUN   (implicit invocation of LINKNGO)
or
/INPUT
/LOAD LINKNGO
/UPSI 1   (tells LINKNGO to read from job stream)
/INCLUDE name
/ENDRUN
```

The '/INCLUDEd' member can have the following formats:

```
 (object deck)
or
 (object deck)
/DATA
(data)
```

OBJECT Utility

The OBJECT utility is primarily used to transfer object programs and LINKNGO control statements from the job stream to the punch area, from where they are loaded into storage for execution.

OBJECT behaves somewhat like a compiler in that it places object programs in the punch area and sets a flag. This flag causes the LINKNGO program to be automatically invoked at the end of the job stream, or whenever a /DATA statement is encountered.

OBJECT may also be used to transfer data from the job stream to the punch area. Setting the NOGO option prevents the LINKNGO program from being invoked if it is not needed.

When UPSI-0 is set on, the OBJECT program transfers the contents of the job stream to the terminal, rather than to the punch area. It thus may be used as an 80-80 list program.

When the NODECK option is set, OBJECT will not write any data to the punch area.

Examples:

1. Cause a subsequent execution of LINKNGO to produce a CSECT map by writing ACTION MAP to the punch area prior to a compilation.

```
/LOAD OBJECT
 ACTION MAP
/LOAD VFORTRAN
 (FORTRAN Source Program)
/DATA
 (Input data records)
```

2. Cause LINKNGO to load and run a module from a VSE library.

```
/LOAD OBJECT
 INCLUDE  name  (name in VSE library)
/DATA
 (input data for execution)
```

3. Cause LINKNGO to load and run an object program stored in the VSE/ICCF library.

```
/LOAD OBJECT
/INCLUDE   name  (name of library member)
/DATA
 (input data for execution)
```

4. Write data to the punch area but prevent LINKNGO from being invoked.

```
/LOAD OBJECT
/OPTION NOGO
 CARD 1
 CARD 2
 CARD 3
 (etc.)
```

5. Write 80 character records from the library (members named MEM1, MEM2, MEM3) to the terminal.
```
/LOAD OBJECT
/OPTION NOGO
/UPSI 1
/INCLUDE MEM1
/INCLUDE MEM2
/INCLUDE MEM3
```

Subroutines

Two subroutines have been provided with VSE/ICCF, DTSSNAP and DTSPGMCK, although you may also be given other subroutines by the VSE/ICCF administrator.

DTSSNAP Subroutine

This subroutine can be used to display a portion of your storage area in storage dump format. This subroutine can be used in assembler, COBOL, FORTRAN and PL/I via the CALL statement.

This subroutine can be called anywhere in your program to obtain a hexadecimal and character display of main storage areas and general registers (except 0 and 1). The subroutine requires two parameters: the labels of the beginning and ending areas to be displayed.

Examples:

Assembler:

```
/LOAD ASSEMBLY
MYPROC     CSECT
           BALR  5,0
           USING *,5
           CALL  DTSSNAP,(TABLE,TABLND)
           EOJ
TABLE      DC    XL256'00'
TABLND     EQU   *
           END
```

COBOL:

```
/LOAD FCOBOL
      ID  DIVISION.
      PROGRAM-ID. MYPROG.
      ENVIRONMENT DIVISION.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      01  TABLE.
          02 TABLEA PIC X(6) OCCURS 30.
      01  TABLEB.
      PROCEDURE DIVISION.
          CALL DTSSNAP USING TABLE TABLEB.
          STOP RUN.
```

DTSPGMCK Subroutine

This subroutine, if called at the beginning of a program, causes a dump of general purpose registers 2 through 13 and the main storage area on the terminal if a program check interrupt occurs.

The dump will be displayed in both character and hexadecimal formats. The main storage area displayed will be the area near where the program check occurred.

The subroutine should only be issued once per program. If the routine is run while some other program check facility (such as the COBOL debug facilities or the FORTRAN debug facilities) is active, the effect of the prior facility will be lost. Generally speaking, the debug facilities inherent in COBOL, FORTRAN, PL/I and RPG II are more useful than this routine. However, this routine can be very useful if you are programming in assembler language.

Example:

```
/LOAD ASSEMBLY
 TESTPRG   CSECT
           BALR 5,0
           USING *,5
           CALL DTSPGMCK
```

Chapter 9. Job Entry Considerations

General Considerations

This chapter provides you with certain basic information on building of job streams and on the execution of jobs in an interactive system.

A section is provided for each language supported. In these sections, any unique requirements for compiling and executing a given programming language will be noted.

Virtual Storage

Virtual storage is a concept whereby frequently used segments of a computer's addressable storage are kept within the actual machine storage of the computer whereas less frequently used segments are saved on direct access (disk) storage.

The advantage of virtual storage is that it can make the overall storage availability of a given computer system appear much larger than the availability of machine storage.

Interactive Partitions

When an /EXEC or /RUN command has been successfully processed, the system attempts to find an available virtual storage area into which to schedule the job. This virtual storage area is referred to as an 'interactive partition' since it appears to have the same characteristics as a normal VSE partition. It has its own communication region, its own work files, its own GETVIS/FREEVIS area (special feature), and its own storage protection key. The storage protection key protects the programs of one interactive partition from accidental destruction by programs in another interactive partition.

GETVIS/FREEVIS

A certain amount of each interactive partition is reserved as GETVIS space. The amount normally reserved is 48K, which is the minimum. If the partition size is 128K (K=1024), 80K will be reserved as the main portion of the interactive partition. Note that some programs and compilers need more than the default 48K of GETVIS space. It depends on how many modules are loaded and the size of the I/O areas and tables in the GETVIS area.

You can alter the default GETVIS/FREEVIS allocation by use of the GETVIS operand on the /OPTION statement. The maximum amount of GETVIS space that you can allocate is your interactive partition size less 20K.

Time-Sliced Execution

Once a job has been successfully scheduled into an interactive partition, it is made eligible for execution. When a job begins execution, it continues to run until one of the following events occurs:

1. A conversational read is encountered.

2. The print area is full or has been forced (/FORCE statement).

3. The job terminates.

4. The interactive partition's time slice elapses.

If a conversational read is encountered, the background job execution is suspended, any print lines in the print area are written on the terminal and the request for conversational input ('?' on 2740/3767 or 'ENTER DATA' on 3270) is displayed. After you have entered the conversational input, the job is again made eligible for execution.

If the print area becomes full, the background job execution is suspended until all print lines have been transferred to the terminal, at which point the job again becomes eligible for execution.

If the job terminates, the interactive partition becomes available for other users, and the remainder of the print output is transferred to the terminal.

When the time slice elapses, execution is suspended but the job is made eligible for execution the next time a time slice becomes available.

Job Streams

Job streams as previously defined comprise job entry statements, source programs and input data. Optionally, the source programs and/or data need not be a part of the physical job stream: they can be included by the /INCLUDE job entry statement. Job entry statements can also be specified in members which are included by /INCLUDE statements.

Job Entry Commands

Two commands can be used to initiate execution of background jobs: /RUN (or /ENDRUN) and /EXEC. If /RUN is used, the job stream or an /INCLUDE referencing the job stream must be in the input area. If /EXEC is used, the job stream (or command list) must be a library member.

The Input Area

This is a temporary disk work space into which you can enter job entry statements, programs and data. See "Temporary Storage Areas" on page 1-8 for a more complete description of the input area.

The Punch Area

The punch area is a temporary disk work space into which the language compilers write their object program output. You can also place 80 column data in the punch area (unit SYSPCH/SYS007 defined below) and read it later. Data in the punch area can be saved in your library by executing the STORE macro command or by inserting the data into the input area and saving it (see the example under /SAVE command). See "Temporary Storage Areas" on page 1-8 for a more complete description of the punch area.

Input/Output

You can obtain your input from the job stream, from the terminal in a conversational manner, from a library member, from a previously written temporary disk file or from a permanent disk file. The output can be written to the terminal, to the punch area, to the print area, to a VSE/ICCF member, to a temporary disk work file or to a permanent disk file.

The choice of input/output type depends on your needs. The type of input/output in most programming languages is decided by how you specify file names and/or VSE 'SYS' unit designations.

The following input/output choices are available to all users:

SYSIPT	Indicates that 80-character records are to be read from the job stream. The data is in the job stream at the logical point where it is to be read. For programs like language compilers, this point would be after the /LOAD or other job entry statements. For your own program compilations and executions this point would be after the /DATA statement. Note that if INCON is specified on an /OPTION or /DATA statement, the input request is converted to a conversational read from the terminal. When defining this unit in a program, it should always be set up as if it were an actual card reader. In VS FORTRAN, SYSIPT corresponds to Unit 5. In VS BASIC, SYSIPT corresponds to the INPUT statement.
SYSLST	Indicates that print lines are to be directed to the terminal for output. In VS FORTRAN, SYSLST corresponds to Unit 6. In VS BASIC, SYSLST corresponds to the PRINT statement. In other programming languages, always define this unit as if it were an actual printer. All line spacing will be suppressed and only single-spacing will be displayed.
SYSPCH	Indicates that 80-character records are to be written to the punch area. Language compilers use this as the unit designation on which the object program is written. You can also use SYSPCH (the punch area) for output to store it later in your library. In VS FORTRAN, SYSPCH is referenced as unit number 7. In other programming languages, always define this unit as if it were an actual punch.
SYSLOG	Indicates input from or output to the terminal. This unit designation can be used for creation of conversational programs. Any read to SYSLOG will be translated into an input request from the terminal while any write to SYSLOG will be treated as a write to the terminal. SYSLOG input can be up to 256 bytes and output can be up to 156 bytes. If a programming language does not permit input on SYSLOG, SYSIPT with the INCON option may be substituted.
SYS000	Is a temporary disk file on which you can record information to be read in a later job step. The file should be defined within your program as a disk file residing on a disk drive. The choice of drive type depends on the disk storage type which the installation is using. Records written or read on SYS000 can be of any size permitted on the device type specified. The file type must be sequential. Since disk files are identified not by 'SYS' number but by 'file name', the file name IJSYS00 must be used to reference the file.
SYS001 SYS002	These are temporary disk files with all the same attributes as SYS000 defined above. Their 'file names' respectively are IJSYS01, IJSYS02 and IJSYS03.

SYS003	The differences between these units/files and SYS000 defined above are: these file areas are used as compiler work areas (but not in VS FORTRAN). They must not be used to pass data from one job step to another if a compiler step intervenes. (SYS003 is not used as a work file by PLIOPT or RPG II.)
SYS004	This is a temporary disk file with all the same attributes as SYS001, 2 and 3. The 'file name' associated with this unit is IJSYS04. The difference between SYS004 and SYS001, 2, 3 is that SYS004 is only used as a work file by the ANS COBOL compiler. None of the other compilers use SYS004 as an intermediate work file.

Note: *If the installation does not support preallocated work files for IJSYS00, 1, 2, 3, and 4, you must specify /FILE information in your job streams to cause these files to be allocated and made available for your use. All work files can be defined as follows:*

/file name=ijsys0n,space=20

where 'n' is specified as appropriate. The amount of space can be changed according to your needs.

SYS005	Card read (job stream) input. SYS005 can be used in place of SYSIPT (defined above) in programming languages which do not permit reference to SYSIPT. SYS005 has all the same characteristics as SYSIPT. However, you should check whether your installation was tailored to have a unit (other than SYS005) for the programmer unit for job stream input.
SYS006	Printer (terminal) output. SYS006 can be used in place of SYSLST (defined above) in programming languages which do not permit reference to SYSLST. SYS006 has all the same characteristics as SYSLST. However, you should check whether your installation was tailored to have a unit other than SYS006 for the programmer unit for print (terminal) output. All line spacing will be suppressed and only single line spacing will be displayed.
SYS007	Punch area output. SYS007 can be used in place of SYSPCH (defined above) in programming languages which do not permit reference to SYSPCH. SYS007 has all the same characteristics as SYSPCH. However, you should check whether your installation was tailored to have a unit other than SYS007 for the programmer unit for punch (terminal) output.
SYS008	Punch area input. SYS008 is used for directly reading the contents of the punch area. This unit indicates that the 80-character records in the punch area resulting from a previous job or job step are to be read. The LINKNGO program uses this unit number as the source of its object program input. When defining this unit in a program, it should always be set up as if it were an actual card reader. (Note the contents of the punch area may also be read on SYSIPT/SYS005 via an /INCLUDE $$PUNCH statement.) You should check whether your installation was tailored to have a unit other than SYS008 for the programmer unit for reading the punch area.
SYS009	Terminal input or output. SYS009 can be used in place of SYSLOG (defined above) in programming languages which do not permit reference to SYSLOG. SYS009 has all the same characteristics as SYSLOG. You should check whether your installation was tailored to have a unit other than SYS009 for the programmer unit for accessing SYSLOG.

Permanent Disk Files

You can access permanent VSE disk files if the installation has provided disk label information (DLBL/EXTENT for the files) in the job stream which starts VSE/ICCF, or if you include label information for the files in your job stream (/FILE statement) which accesses the files.

You may specify file information (via the /FILE statement) for normal VSE files if the following conditions are met:

1. Your user profile allows you to specify VSE file information.
2. The file being requested is not ISAM.
3. The file consists only of a single extent.

If these criteria are not met, you can only access the requested file if in fact the label information (DLBL/EXTENT) for the file had been specified in the VSE/ICCF startup job stream or in a standard label area. For example, the only way to access an ISAM file from a program in a VSE/ICCF interactive partition is to have the DLBL/EXTENT statements for the file in the VSE/ICCF startup job stream or in a standard label area.

If permanent disk files are to be referenced in your programs, keep in mind the following points:

1. The name you specify for the file when you define it in your program must be the same as the 'file name' on the DLBL statement or the NAME= parameter on the /FILE statement.
2. VSE/ICCF provides no facilities for protecting these files from simultaneous updating. You must therefore provide this protection.
3. Permanent data sets can be organized as sequential, direct, VSE/VSAM or indexed sequential and accessed according to standard VSE access method conventions.

Overall Job Restrictions

Interactive processing requires that certain limitations be observed. For example, timing devices should not be built into programs. The setting of timer intervals in background jobs is prohibited.

Tape file processing is not generally supported in background jobs.

When defining the SYSIPT, SYSLST, SYSPCH, SYS005, SYS006, SYS007, SYS008, SYS009 unit record type files, always specify them as unblocked, fixed length, unlabeled and single buffered where explicit definition is required.

The assembly (or compilation) and execution of multi-phase programs is not supported because the LINKNGO utility will not support them. However, a multi-phase program may be compiled and linked (via the SUBMIT to VSE/POWER procedure) to a VSE library. Once in the VSE library, the multi-phase program can be loaded and run in VSE/ICCF by specifying its name on the /LOAD job entry statement.

The format of the print output produced by the RELIST macro may be different from the format you would get if you run the same program in a VSE batch partition and send the print output directly to a printer or to a VSE/POWER print queue.

For more details on restrictions in interactive partitions see *Appendix A. Interactive Job Restrictions.*

Job Termination

When a job either terminates normally, or is canceled by VSE or by VSE/ICCF, the following message is printed at the end of the job's terminal output:

```
**** JOB TERMINATED -VSE/ICCF RC xx, EXEC RC xxxx, NORMAL EOJ
                                                             or
**** JOB TERMINATED - ICCF RC xx, JOB CANCELED
```

A VSE/ICCF RC of '00' is the normal job ending code. The VSE/ICCF return codes indicated by 'xx' in the message are described in detail in *VSE/ICCF Messages*, SC33-6205. The EXEC RC is the job execution return code as set and documented by the program that was called.

A VSE/ICCF return code of '08' will usually be preceded by a VSE message which more fully explains the reason for job termination. These codes are described in *VSE/ICCF Messages* SC33-6205.

An interactive partition is canceled if the job executing in it exceeds the time limit set for the partition. If, however, terminal I/O is outstanding when this partition limit is reached, then the partition will not be canceled immediately. It will be canceled either after the I/O is done, or when the terminal time-out occurs -- depending on which event happens first.

Assembler and Compiler Considerations

This section contains information pertinent to the use of the assembler and compilers which are supported by VSE/ICCF.

The following is a list of compilers explicitly supported. Your particular installation may have chosen not to implement some of these compilers. Or, it may have chosen to install compilers which are not in this list. The information concerning installation supported compilers must be obtained from the installation staff.

```
1.  Assembler -                    /LOAD ASSEMBLY
2.  VS BASIC For DOS/VS -          /LOAD BASIC
3.  DOS/VS FULL ANS COBOL -        /LOAD FCOBOL
4.  VS FORTRAN                     /LOAD VFORTRAN
5.  PL/I Optimizing Compiler -     /LOAD PLIOPT
6.  DOS/VS RPG II -                /LOAD RPGII
```

The following pages contain information and restrictions which pertain to each of the individual compilers.

Interprogram Linkage

The LINKNGO utility combines program segments which have been produced by different language compilers (except BASIC) into an executable program. However, standard IBM VSE linkage conventions must be maintained, and interprogram linkage must be supported by the given compiler.

The following job entry statements can be used to compile a PL/I program that calls an assembler subroutine which is to be assembled in the same job stream.

```
/LOAD PLIOPT
   .
   .
 CALL SUBRTN
   .
   .
/LOAD ASSEMBLY
 SUBRTN START 0
   .
   .
        BR 14
/DATA
   .
   .
(data records, if any)
```

In addition, the program or subprogram to be loaded with other programs or subprograms may appear in the job stream in object (already compiled) form.

For example, if SUBRTN alone had been compiled and its object program stored in the library under the name SUBOBJ, the job stream might look as follows:

```
/LOAD PLIOPT
   .
   .
 CALL SUBRTN
   .
   .
/LOAD OBJECT             (optional)
/INCLUDE SUBOBJ
/DATA
   .
   .
  (data records)
```

It is also possible that all programs and subprograms have been separately compiled and stored in the library. The execution job stream would then be:

```
/LOAD OBJECT            (optional)
/INCLUDE PLIOPOBJ
/INCLUDE SUBOBJ
/DATA
  .
  .
 (data records)
```

The main program should always be the first source or object program in (or included in) the job stream, so that the proper transfer address is obtained.

Note in the examples above that the /LOAD OBJECT is optional. The OBJECT utility causes the object programs to be transferred to the punch area for loading by LINKNGO. If /LOAD OBJECT is omitted and an object program (except BASIC) is encountered in the job stream, the OBJECT utility is invoked automatically.

Assembler Considerations

VSE/ICCF supports the assembler for program creation and execution. In addition, subprograms or subroutines may be written in assembler language and entered from programs written in COBOL, FORTRAN, PL/I, or RPG II through the standard VSE CALL linkage conventions.

The combining of subroutines with a main program is performed by the LINKNGO utility program. LINKNGO can combine object modules in the punch area with object modules from the library. Unresolved external references are resolved from a VSE library.

Notes and Restrictions:

1. At program invocation the VSE register conventions are obeyed.

   R0 : Load point of loaded phase.
   R1 : Dependent on the /LOAD job entry statement:
   If parameter was passed: pointer to parameter fields
   Else: entry point of loaded phase.
   R13: Pointer to 18 fullword save area.
   R14: Return point to VSE/ICCF.
   R15: Entry point of loaded phase.

2. Because the LINKNGO program only produces single phase programs, the LOAD and FETCH macros should not be used unless you are loading or fetching a phase from a VSE library.

3. Programs issuing the following macro functions are not supported in interactive partitions.

```
ATTACH                JOBCOM           STXIT(MR|TT)
DETACH                PAGEIN           TESTT
DEQ                   PFIX             TPIN
ENQ                   PFREE            TPOUT
EXCP ccbname,REAL     POST             VIRTAD
EXIT(AB|MR|TT)        REALAD           WAITF
FCEPGOUT              SETPFA           WAITM
FREE                  SETT
```

4. The STXIT OC, STXIT PC and STXIT AB macro facilities can be used to provide for operator attention, program check or abnormal termination intercept.

5. End of file on SYSIPT or SYS005 (or equivalent unit) can be recognized by testing for '/* ' in columns 1 - 3 of the read input area or by testing the unit exception bit in the CCB. When using standard GET/PUT/DTFCD card read facilities, you do not have to take any extra precautions to recognize end of file.

6. If invalid CCWs are issued by the program they are ignored.

Example:

The following assembler program will read statements, write them onto a temporary disk file, read them back in and list them on SYSLST (that is, on the terminal). As each record is read the second time, a subroutine is entered to verify the validity of the data.

```
*READY
/input
/load assembly
/include workfile      (if no preallocated work files)
Mainpgm   csect
          balr   5,0                    load base reg
          using  *,5                    declare base reg
          open   filout                 open output file
loopa     excp   rdccb                  read a statement
          wait   rdccb                  wait on completion
          clc    r(3),=c'/* '           check for last statement
          be     endcard                go to file end routine
          mvc    0(80,2),r              move image to filebuffer
          put    filout                 write to output file
          b      loopa                  loop through all input
endcard   close  filout                 close output file
          open   filin                  open the input file
loopb     get    filin                  read record
          la     13,savearea            savearea pointer
          lr     1,2                    dataarea pointer
          call   subpgm                 enter the subroutine
          ltr    1,1                    test error condition
          bz     cardok                 bypass message if ok
          mvc    p+85(12),=c'invalid statement'
cardok    mvc    p(80),0(2)             move image to print
          excp   prccb                  write to syslst
          wait   prccb                  wait i/o completion
          mvc    p,=cl120' '            clear print area
          b      loopb                  process next record
enddisk   close  filin                  close input file
          eoj
savearea  ds     9d                     save area
p         dc     cl120' '               print area
r         dc     cl80' '                input area
rdccb     ccb    sysipt,rdccw           card read ccb
prccb     ccb    syslst,prccw           printer ccb
rdccw     ccw    2,r,0,80               read command
prccw     ccw    9,p,0,120              print command
ioa       ds     cl408                  disk i/o area1
iob       ds     cl408                  disk i/o area2
filin     dtfsd  ioarea1=ioa,ioarea2=iob,ioreg=(2),typefle=input,  x
                 blksize=400,recsize=80,device=3330,               x
                 eofaddr=enddisk,recform=fixblk
          org    filin+22
          dc     c'ijsys00'             set filename to ijsys00
          org
filout    dtfsd  ioarea1=ioa,ioarea2=iob,ioreg=(2),typefle=output, x
                 blksize=408,recsize=80,device=3330,recform=fixblk
          org    filout+22
          dc     c'ijsys00'             set filename to ijsys00
          org
          end
```

```
/load assembly
/include workfile
subpgm    csect
*this subroutine is called by the main program to validate data
          save   (14,12)             save calling program regs
          lr     5,15                set reg 5 as base
          using  subpgm,5            declare base
          la     4,5                 5 is max no of data columns
          lr     3,1                 1 contains data area pointer
loop      cli    0(3),c'0'           is column numeric
          bl     errext              no - error exit
          la     3,1(3)              bump to next byte
          bct    4,loop              loop till all done
goodex    lm     14,12,12(13)        restore entry regs
          sr     1,1                 clear reg 1 indicates valid
*                                    card condition
          br     14                  return to caller
errext    return (14,12)             return to caller
          end
/data
/file name=IJSYS00,space=1 (if no preallocated IJSYS00)
03754     this is a valid statement
76843     this is a valid statement
42d68     this is an invalid statement
12345     this is a valid statement
/end
*READY
/run
*RUN REQUEST SCHEDULED
```

Note in the above example that both FILIN and FILOUT refer to the same file area. One DTFSD is for output, the other for input. The temporary file IJSYS00 is used for storing and retrieving the data. Because one cannot name both DTFs IJSYS00 nor use one DTF for both input and output, an ORG to filename+22 appears after each DTF. This is the area in the DTF macro expansion where the file name is kept. After the ORG, a DC C'IJSYS00' appears. This causes the name in the DTF area to actually be IJSYS00 for both files. When each file is opened, it will be set up to print to the SYS000 (IJSYS00) temporary file area.

Also, in the above example, if the subroutine SUBPGM had been assembled separately and the resulting object program stored in the library under the name SUBOBJ, the job stream could have been specified as follows:

```
/LOAD ASSEMBLY
 (source statements for MAINPGM as above)
/LOAD OBJECT         (this is optional)
/INCLUDE SUBOBJ
/DATA
 (data records as above)
```

The /LOAD OBJECT is optional since it is automatically invoked if an object program is encountered in the job stream.

Full Screen Macro (DTSWRTRD)

The DTSWRTRD macro enables your assembler programs in interactive partitions to manipulate the screen of the IBM 3270 Display System. For example, it allows you to dynamically construct and display screens, or to support the selector pen. You must provide the output data stream in an output buffer and analyze the input data stream that VSE/ICCF returns to the input buffer. The output data stream that you build and send to the terminal is not modified by VSE/ICCF. Also, VSE/ICCF does not check or change the input data before transmitting it to your program. (The input data only contains those data fields that have their modified tags set to 'on').

All parameters are optional. A full screen write does not have to be followed by a read. However, where this occurs, an implicit wait is performed. The macro has the following format:

```
[name]    DTSWRTRD
          [WRTBUF = {name1|(r1)|(S,name1)}]
          [,WRTBUFL={integer|selfdefiningterm|(r2)|(S,name2)}]
          [,RDBUF={name3|(r3)|(S,name)}]
          [,RDBUFL={integer|selfdefiningterm|(r4)|(S,name4)}]
          [,WRTTYPE={WF|TS|TI|RS|RI|name5|(r5)|(S,name5)}]
          [,MFG={name6|(r6)|(S,name6)}]
          [,TYPE={EXECUTE|DSECT}]
```

WRTBUF specifies the address of the write buffer. The contents of the write buffer can be structured in two ways, depending on the write request which you specify in the WRTTYPE parameter. In both cases the data stream is transferred unchanged to the display station.

WRTTYPE=RS|RI|TS|TI
can be specified regardless of whether your terminal supports the IBM 3270 Extended Data Stream Feature or not (use the DTSSCRN macro to find out about the characteristics of your terminal). The first byte in the write buffer must be the write control character (WCC). If your terminal supports the IBM 3270 Extended Data Stream Feature, the following bytes can contain the 3270 extended data stream; otherwise they should contain the non-extended 3270 data stream. The extended 3270 data stream, which comprises the non-extended 3270 data stream, is described in detail in *IBM 3270 Information Display System, Data Stream, Programmer's Reference.*

WRTTYPE=WF
can be specified only, if your terminal supports the IBM 3270 Extended Data Stream Feature. In this case, no WCC may occur at the beginning of the write buffer and the data stream has to be a collection of structured fields. For each one of these structured fields, specify a length greater zero. The format of structured fields in outbound data streams is described in detail in *IBM 3270 Information Display System, Data Stream, Programmer's Reference.* Only three values can be specified in the identification field of the structured field:

ID	**Name**
X'01'	Read Partition
X'06'	Load Programmed Symbols
X'09'	Set Reply Mode

If you have a structured field with the ID 'Read Partition', make sure that:

- This structured field comes last in your data stream.
- You specify 'Query' (X'02') as Type for this structured field.

The output area may be located in the interactive partition issuing the DTSWRTRD macro or in the SVA. If WRTBUF is not specified, it is assumed to have been established previously in the area pointed to by the MFG parameter, which is required in this case.

WRTBUFL specifies the length of the write buffer. The minimum is 1 byte, the maximum 32700 bytes. For performance reasons, a value less than the device dependent TIOA size should be specified. We recommend that, depending on the device, you specify the following buffer lengths:

- 2000 bytes for the IBM 3270 Model 2
- 2660 bytes for the IBM 3270 Model 3
- 3560 bytes for the IBM 3270 Model 4
- 3650 bytes for the IBM 3278 Model 5 wide screen

The program can investigate the screen size of the terminal with which it is communicating by issuing the DTSSCRN macro. If WRTBUFL is not specified, it is assumed to have been established previously in the area pointed to by the MFG parameter, which is required in this case.

RDBUF specifies the address of the read buffer. The contents of the read buffer can be structured in two ways, depending on the write request which you specified in the WRTTYPE parameter.

WRTTYPE=RS|RI|TS|TI

the buffer starts with a 2-byte length field which contains the length of the data stream. If this length is greater than RDBUFL, the data stream is truncated to the length of RDBUFL - 2. The length field is followed by the inbound 3270 data stream. The inbound 3270 data stream, is described in detail in *IBM 3270 Information Display System, Data Stream, Programmer's Reference*. All input data (AID, cursor address, buffer address, data ....), with the exception of the cursor address, is transferred unchanged from the terminal into the input area following the length field. The cursor address, however, is transformed into binary notation before it is placed into the read buffer (the first screen byte, for example, has the cursor address X'0'). The AID's transferred are the PF keys, the CLEAR key, the ENTER key, and the selector pen attention. All PA keys are reserved for use by VSE/ICCF. If a PA key is pressed, the 2-byte length field contains X'0'.

WRTTYPE=WF with Read Partition Query

the buffer starts with a 2-byte length field which contains the length of the data stream. If this length is greater than RDBUFL, the data stream is truncated to the length of RDBUFL - 2. The length field is followed by a one-byte Attention Identifier field, which has a value of X'88'. The Attention Identifier field is followed by one or more Query Reply structured fields. The various formats of the Query Reply structured field are described in detail in *IBM 3270 Information Display System, Data Stream, Programmer's Reference.*

If RDBUFL is not 0 and RDBUF is not specified, it is assumed to have been established previously in the area pointed to by the MFG parameter, which is required in this case.

RDBUFL	specifies the length of the read buffer (minimum 2, maximum 32K - 1(32767)), including 2 bytes for the length field. If more data arrives, it is truncated to RDBUFL - 2 bytes.

A value of 0 for RDBUFL indicates WRITE without READ, in which case the RDBUF address is ignored. In case WRTTYPE=WF with ID=Read Partition was specified, the minimum length of the read buffer is 1 byte. Number and length of the returned Query Reply structured fields determine the necessary size of the read buffer.

In case WRTTYPE=RS|RI|TS|TI was specified, the actual maximum length of an input data stream that a program can read depends on the screen size of the terminal which the program is communicating with. For terminals supported by VSE/ICCF the maximum input data stream is twice the screen size of the terminal. The program can investigate the screen size of the terminal with which it is communicating via the DTSSCRN macro. If RDBUFL is not specified, it is assumed to have been established previously in the area pointed to by the MFG parameter, which is required in this case.

WRTTYPE	specifies whether the previous contents of the screen are to be erased before the write buffer is written. It also specifies whether VSE/ICCF is to save and restore the screen (for example when a message arrives and destroys the contents of the screen), or whether your program will be notified via a return code that the screen has been destroyed and must be rebuilt if it is still needed. WRTTYPE=TS causes the ERASE WRITE command to be issued. WRTTYPE=RS also causes the ERASE WRITE command to be issued except that, additionally, a return code is placed in register 15 indicating to your program that the screen must be rebuilt if it is still needed. WRTTYPE=TI causes the WRITE command to be issued without a previous erase. WRTTYPE=RI also causes the WRITE command to be issued without a previous erase except that, additionally, a return code is placed in R15 indicating to your program that the screen must be rebuilt, if it is still needed. WRTTYPE=WF causes the WRITE STRUCTURED FIELD command to be issued without a previous erase. If the content of the screen has been destroyed, a return code is placed in R15 indicating to your program that the screen must be rebuilt, if it is still needed. For all other specifications the address value must point to a field containing the character string TS, TI, RS or RI.

Notes:

1. *RS and RI should be used whenever possible, since TS and TI use more storage, and they also increase message traffic, especially on remote lines.*

2. *A WRITE/ERASE is forced when:*

   - *the CLEAR key has been pressed*
   - *the program issues the first full screen write request, or*
   - *the program changes the WRTTYPE from RS/RI to TS/TI or vice versa.*

MFG	allows the address of a parameter list to be specified. If not specified, the parameter list is generated inline with the macro call, in which case modification and reuse of the parameter list is not possible. If specified, all other parameters are entered into the pertinent slots of the list that MFG is pointing to. This way parameters can be overwritten or reused if specified in a previous macro call to DTSWRTRD. The length of the parameter list can be determined with the DTSWRTRD macro with the TYPE=DSECT parameter. The area in which the parameter list is to be built must be cleared to binary zeros before the first macro is entered.
TYPE	specifies the execute, or DSECT, form of the macro. EXECUTE specifies actual execution of the terminal I/O function, which is the default. DSECT specifies that a DSECT describing the parameter list for DTSWRTRD is to be generated. The name of the DSECT is taken from the macro invocation. If no name is specified, the DSECT name defaults to DTSWRDD.

Some examples for possible operand specifications:

```
WRTBUF=BUFF              buffer address
WRTBUF=(2)               register containing the buffer address
WRTBUF=(S,BUFF)          S-type constant for relocating purposes
WRTBUFL=2000             absolute length value
WRTBUFL=L'BUFF           implicit length constant
WRTBUFL=LEN              length equate
WRTBUFL=(S,BUFL)         S-type constant pointing to length value
BUFF DC ...
LEN EQU ...
BUFL DC H(..)
```

If TYPE=DSECT is specified, the following DSECT is produced:

```
DTSWRDD   DSECT ,
DTSWRDO   DS    F       address of output area
DTSWRDOL  DS    H       length of output data stream
DTSWRDOT  DS    CL2     type of write (TS, TI, RS, RI, WF)
DTSWRDI   DS    F       address of input area
DTSWRDIL  DS    H       length of input area
DTSWRDIT  DS    CL2     type of read ('RD')
DTSWRDND  DS    0C      end of table label
DTSWRDL   EQU   *-DTSWRDD
DTSWRDNR  EQU   0       normal return
DTSWRDLI  EQU   4       inp len incorrect short on storage
DTSWRDSR  EQU   8       program has to restore screen
DTSWRDSH  EQU   12      shutdown warning issued
DTSWRDDI  EQU   16      invalid output data stream
DTSWRDNS  EQU   20      EDS not supported
DTSWRDPI  EQU   28      parameter list invalid
DTSWRDDE  EQU   32      data stream error
DTSWRDMS  EQU   64      message suppressed
```

Register usage:

R0	work register
R1	contains the address of the parameter list.
R15	return code register

Return code 0
: Normal return.

Return code 8
: Occurs only with WRTTYPE=RS or RI and indicates that the program must restore the screen, if necessary.

Return code 12
: Indicates that the VSE/ICCF operator command /WARN has been issued. Register 15 will contain this code each time DTSWRTRD is issued between /WARN and /WARN RESET or shutdown.

Return code 16
: Indicates one of the following cases:

  - The structured field in the data stream which you submitted had an ID other than Load Programmed Symbol, Set Reply Mode, or Read Partition.
  - The Read Partition structured field was not the last in your outbound data stream.
  - The length of the data stream that was submitted with WRTTYPE=WF was different from the length of the write buffer.
  - The length of one of the structured fields in your outbound data stream has not been specified correctly.

Return code 20
: Occurs only with WRTTYPE=WF and indicates that your terminal does not support the IBM 3270 Extended Data Stream Feature.

Return code 28
: Invalid parameter list specified by user.

Return code 32:
: An error was found in the Write Structured Field request.

Return code 64
: Occurs only if WRTTYPE=TI|TS|WF and if the terminal supports the IBM 3270 Extended Data Stream Feature. It indicates that the display of a message, which is pending for the user, is suppressed, even though the user had requested automatic message display.

Restrictions and Special Considerations

1. A program that uses the full screen service must not be called from inside a procedure in such a way that it needs a second interactive partition. You can, however, call the program from a procedure if the MULTEX option is off or by issuing /PEND prior to the /RUN or /EXEC request for that program.

2. When a program issues DTSWRTRD:

   - The interpretation of VSE/ICCF control characters, like line-end, is skipped.

- PF key setting as well as scrolling through the print output from the program is skipped.
- Special screen settings, like number of input lines, screen size specifications, etc. are ignored.
- Hardcopy mode is ignored.
- Continuous output mode has no effect.
- No translation of input and output data is performed.
- The messages *ENTER DATA? and *PARTIAL END PRINT are suppressed.
- No scale line appears on the screen.
- PA2 key causes /CANCEL AB to be processed.
- PA1 key causes the OC exit to be entered.

As soon as the program gets control back from DTSWRTRD, the above settings are in effect again as they were before the program issued DTSWRTRD. After DTSWRTRD has been processed, VSE/ICCF does not touch the screen until the program issues a regular SYSLOG or SYSLST request. This means that, between successive DTSWRTRD requests, VSE/ICCF does not send a scale line to the screen, nor does it manipulate control settings of the terminal, such as keyboard restore.

3. The message

   *PARTIAL END PRINT BEFORE FULL SCREEN WRITE

   is issued immediately before the DTSWRTRD output data is displayed on the screen (in case there is other print output from SYSLOG and/or SYSLST in front of DTSWRTRD). Either press ENTER to get the DTSWRTRD output data, or issue any command that can be entered during execution spool print mode.

4. If light pen selection and attention is used, a program must always be prepared to accept the input from the terminal via ENTER, which means that it also receives the data of the modified fields.

5. A WRITE/ERASE will be forced under the following three circumstances:

   - when the CLEAR key is pressed,
   - when the program issues the first full screen write request,
   - when the program changes the WRITTYPE from RS/RI to TS/TI, or vice versa.

6. If the automatic message display feature is set on and a message is pending for the user,it depends on the value specified for the WRTTYPE parameter and on the terminal characteristics whether the message is displayed:

   - If the terminal does not support the IBM 3270 Extended Data Stream Feature, the message is displayed immediately.
   - If the terminal supports the IBM 3270 Extended Data Stream Feature and if *WRTTYPE=TI|TS|WF*, the message is not displayed and return code 64 is passed to the program.

- If the terminal supports the IBM 3270 Extended Data Stream Feature and if *WRTTYPE=RI|RS*, the message is displayed and it's the program's responsibility to save and restore the screen.

Macro to Identify Terminal Characteristics (DTSSCRN)

If a program running in an interactive partition communicates with a terminal, it needs to know certain characteristics of that terminal:

- If the program communicates with the terminal via I/O requests to SYSLOG and SYSLST, it needs to know the **logical screen size** which has been defined with the /SET SCREEN command. The logical screen size must be known to the program because it determines the number and the length of output lines which the program generates.
- If the program communicates with the terminal via the DTSWRTRD macro, it needs to know the **physical screen size** (the logical screen size is not considered by the DTSWRTRD macro).

To determine the logical and the physical screen size of a terminal, call the DTSSCRN macro in your program. The DTSSCRN macro returns information on the following terminal characteristics:

- physical screen size characteristics
- logical screen size characteristics
- terminal device type
- Extended Data Stream features

The format of the macro is:

```
[name] DTSSCRN DSECT={YES|NO}
               [,AREA={name|(r)|(S,name)}]
```

AREA specifies the address of an area into which VSE/ICCF will place the terminal characteristics. Define this area with a length of DTSCLEN. If the terminal does not have a device type of 327x or 329x, the whole area except DTSTYPE is set to X'00'.

DSECT=YES causes a dummy section to be generated for the area that will contain the terminal characteristics. The name for this area is taken from the macro specification. If no name was specified, the name defaults to DTSSCRN.

DSECT=NO is assumed if the parameter has not been specified.

The dummy section has the following format:

```
DTSSCRN  DSECT   ,               SCREEN INFORMATION MAP
*                PHYSICAL SCREEN CHARACTERISTICS
DTSPHLNS DS      C                   NO. OF SCREEN LINES
DTSPHCLS DS      C                   NO. OF SCREEN COLUMNS
*                LOGICAL SCREEN CHARACTERISTICS
DTSLOINP DS      C                   NO. OF SCREEN INPUT LINES
DTSLOSOT DS      C                   STARTING LINE FOR OUTPUT
DTSLOOUT DS      C                   NO. OF SCREEN OUTPUT LINES
DTSLOSTR DS      C                   SCREEN STARTING COLUMN
DTSLOEND DS      C                   SCREEN ENDING COLUMN
*                TERMINAL DEVICE TYPE
DTSTYPE  DS      C                   TERMINAL DEVICE TYPE
DTSX3270 EQU     1                   327N, 328N, 3290, OR 5550
DTSX2741 EQU     2                   2741
DTSX2740 EQU     3                   2740,1050,....
DTSXANY  EQU     4                   ANY OTHER TERMINAL
                 3270 EXTENDED DATA STREAM FEATURES
DTSC32EF DS      C                   3270 EXTENDED FEATURES
DTSCEDS  EQU     X'80'               EXT.DATA STREAM SUPPORTED
DTSCCOL  EQU     X'40'               COLOR SUPPORTED
DTSCPSS  EQU     X'20'               PROGRAMMED SYMBOLS SUPPORTED
DTSCHIL  EQU     X'10'               HIGHLIGHT SUPPORTED
DTSCVAL  EQU     X'08'               VALIDATION SUPPORTED
*
DTSC32E2 DS      C                   3270 EXTENDED FEATURES
DTSCFRL  EQU     X'80'               FIELD OUTLINING SUPPORTED
DTSCMIX  EQU     X'40'               MIXED FIELD SUPPORTED
DTSTIOAS DS      H                   TIOA-SIZE FOR SCREEN
DTSSCEND DS      OC                  END OF TABLE LABEL
DTSCLEN  EQU     DTSSCEND-DTSPHLNS   LENGTH OF DSECT
```

The TIOA size (DTSTIOAS) which DTSSCRN returns is not the maximum buffer size (32700) that can be used for a single full screen write to the terminal. For performance reasons, however, we recommend that you specify this size in the WRTBUFL parameter.

Register usage:

R0	work register
R1	address of the terminal characteristics area.

Return codes: None

VS BASIC Considerations

VSE/ICCF supports the DOS/VS version of the VS BASIC compiler. Most of the VS BASIC features are available to you under VSE/ICCF.

The VS BASIC compiler is a single-pass compiler which does not use work files. It is the fastest of all the supported compilers since it is the only one designed to be terminal oriented and to take advantage of virtual storage.

The VS BASIC compiler includes its own loader facility. LINKNGO is never used to load BASIC object programs. For most small programs, it is faster to recompile the source program than to save and then re-run an object deck (although this may be done).

Normally the VS BASIC compiler does not produce an object program on SYSPCH unless specifically told to do so. Instead the source program is compiled and the object program loaded into storage for immediate execution.

The VS BASIC compiler and library routines are reenterable, which means that the same copy of the compiler can be in use by several terminal users at the same time.

Notes and Restrictions:

1. The VS BASIC INPUT statement can be used to read 80-character records from the job stream. The INPUT verb uses the VSE SYSIPT unit as the source of the input data.

2. The VS BASIC INPUT instruction can be used to read terminal input conversationally by specifying the INCON option on the /DATA statement.

3. The first release of the DOS/VS version of VS BASIC caused SYSIPT data to be read in a double buffered, overlapped manner. While this is good for batch execution, it presents difficulties when conversational programs are being created. If your installation has the double buffered SYSIPT version of VS BASIC and if you use the INPUT statement via the /DATA INCON facility to set up a conversational program, you will find that your requests for input (INPUT) are not synchronized with your displays (PRINT) to the terminal. The latest version of VS BASIC allows SNGL to be specified on the RUN statement to force single buffering.

4. The first 80-character record read by VS BASIC should be the RUN command. The VS BASIC RUN command specifies where the program is to be obtained and whether a source listing is to be produced. Normally the command with default options is 'RUN * SNGL'. The SNGL operand causes single buffering of INPUT command input.

   Some typical RUN command formats are:

```
•  RUN * SNGL,NOLIST           to suppress the source listing.
•  RUN * SNGL,NOGO,STORE=*     to suppress execution but to produce an
                                  object program in the punch area.
•  RUN * SNGL,INPUT=*          to read via the INPUT statement.
•  RUN * SNGL,OBJECT,INPUT=*   to run an object program which reads
                                  data from SYSIPT (INPUT statement).
```

5. The VS BASIC Programmer's Guide indicates that the special delimiter ' / ' (space, slash, space) must be used to separate the source programs from the data. The /DATA statement may be used in place of this special delimiter. However, no other job entry statements may follow the /DATA

statement. Any job entry statements pertaining to the execution of the program should follow the /LOAD BASIC statement.

6. If you are using VS BASIC you can define stream oriented files SYS000, SYS001, SYS002, SYS003, and SYS004. Files SYS005 through SYS009 may or may not be available depending on how the installation staff have installed VSE/ICCF. The stream oriented files SYS000 through SYS004 correspond to the files IJSYS00 through IJSYS04 defined in the preceding section. They are temporary files (although they may be retained by copying them to permanent files using the DTSCOPY utility).

   Note that for VS BASIC stream oriented files consist of 120-byte unblocked records.

7. If you are using VS BASIC and you want greater stream oriented file flexibility, you can use the file copy utility program (DTSCOPY). This utility transfers a stream oriented file with any name to one of the temporary file areas (SYS000 through SYS004), from where it can be accessed or updated. The DTSCOPY utility can also be used to transfer the file from the temporary area to the permanent file area. Similarly, you can supply /FILE statements to define the SYS00n files as required.

8. VS BASIC record-oriented files can only be accessed if the computer installation supports VSE/VSAM and if VSE/VSAM files have been set up for access under VS BASIC.

9. The VS BASIC compiler does not use temporary disk work files during compilation. Any temporary file SYS000 through SYS004 can be created and passed to a succeeding step regardless of intervening VS BASIC compilations.

10. If you have set up a VS BASIC sequence of chained programs which are to read data conversationally and then return to the job stream to process the next chained program, you can enter '/DATA NOINCON' following the last conversational read from the program to return SYSIPT reading to the actual job stream containing the program.

Example:

This example reads 80-character records containing three numeric integers. It performs calculations on the input data, displays the input data and finally displays the result of the calculations.

```
*READY
/input
/load basic
run * input=*
010 print 'start of program'
020 input i,j,k
030 l = i + j + k
040 m = i * &sqr2
050 n = j * &pi
060 print i,j,k
070 print l,m,n
080 go to 020
090 end
/data
100,200,21
15,75,42
/end
*READY
/run
*RUN REQUEST SCHEDULED
```

ANS COBOL Considerations

VSE/ICCF supports the DOS/VS FULL ANS COBOL compiler. Most features of the language are supported by VSE/ICCF.

The object programs produced by the compiler can be combined with other COBOL programs or programs written in other languages and loaded into storage for execution by the LINKNGO utility.

Options which control execution of the compiler may be supplied on the /OPTION statement or on the COBOL 'CBL' statement. An option is available on the CBL statement to perform a syntax-only scan of the source program.

Some typical COBOL 'CBL' statement examples are as follows:

- CBL NOSEQ,SYNTAX for SYNTAX only checking
- CBL NOSEQ,FLOW,STATE for debugging a program
- CBL NOSEQ,OPT for optimized compilation

Most ANS COBOL debugging features are supported. However, to use the SYMDMP option, the special work file IJSYS05 must have been defined and the installation must have initiated VSE/ICCF with SYS005 assigned to disk.

The FULL ANS COBOL Compiler contains some very powerful debugging tools which can be of significant help when debugging large, complex programs. See the Programmer's Guide for more information.

Notes and Restrictions:

1. Segmented COBOL programming is not supported when compiling and executing in an interactive partition. Segmented programs must be cataloged to a VSE library via the submit-to-batch facility. They can then be accessed via the /LOAD statement.

2. The COPY facility for predefined COBOL source code is supported.

3. The COBOL ACCEPT statement can be used to read 80-character records from the job stream. Its use corresponds to VSE unit SYSIPT.

4. The COBOL ACCEPT FROM CONSOLE statement can be used to read data from the terminal. This statement should be used when writing conversational programs in COBOL.

5. The DISPLAY statement with or without the UPON CONSOLE phrase will write data lines to the terminal.

6. When defining card read (job stream input) files, the ASSIGN clause should specify SYSxxx-UR-2540R-S where 'xxx' is normally 005 but may be some other installation specified SYS number.

7. When defining print (terminal output) files, the ASSIGN clause should specify SYSxxx-UR-1403-S where 'xxx' is normally 006 but may be some other installation specified SYS number.

8. When defining punch area output files, the ASSIGN clause should specify SYSxxx-UR-2540P-S where 'xxx' is normally 007 but may be some other installation specified SYS number.

9. The data written to the punch area can be read back by defining a file whose ASSIGN clause specifies SYSxxx-UR-2540R-S where 'xxx' is normally 008 but may be some other installation specified SYS number. Alternately, punch data can be read from the input stream by specifying /INCLUDE $$PUNCH in the input stream.

10. Temporary sequential disk work files can be defined. Their ASSIGN clauses should specify SYS00n-DA-XXXX-S-IJSYS0n where 'n' is 0, 1, 2, 3, or 4 and XXXX is the disk device type. Remember that all temporary files except IJSYS00 will be destroyed by the next ANS COBOL compilation. If preallocated work files are available, these files need not be defined via /FILE statements.

Example:

The following example reads 80-character records from the job stream, writes them back to the terminal and also stores them in a disk area (IJSYS00).

```
*READY
/input
/load fcobol
/include workfile     (if no preallocated work files)
     id   division.
     program-id. jactest.
     environment division.
     input-output section.
     file-control.
     select infile assign sys005-ur-2540r-s.
     select outfile assign sys000-da-3330-s-ijsys00.
     select prtfile assign sys006-ur-1403-s.
     data division.
     file section.
     fd   infile label records omitted.
     01   inrec pic x(80).
     fd   prtfile label records omitted.
     01   prtrec pic x(80).
     fd   outfile label records standard block contains 5 records.
     01   outrec pic x(80).
     procedure division.
     open input infile output outfile prtfile.
     mnloop.
          read infile at end go to endjob.
          write prtrec from inrec.
          write outrec from inrec.
          go to mnloop.
     endjob.
          close infile prtfile outfile.
          stop run.
/data
/file name=ijsys00,tracks=2
first record
second record
third record
/end
*READY
/run
*RUN REQUEST SCHEDULED
```

VS FORTRAN Considerations

VSE/ICCF supports the VS FORTRAN compiler. Most of the language features are available to the VSE/ICCF user.

Subroutines written in other languages (for example, assembler) can be called by a FORTRAN program. The FORTRAN program and its subprograms are loaded and linked together using the LINKNGO utility.

Notes and Restrictions:

1. Options controlling the execution of the compiler can be specified in either of the following:

   - The @PROCESS statement of VS FORTRAN.
   - The PARM field of the /LOAD VFORTRAN statement.

   They cannot be specified in the /OPTION job entry statement.

2. Multiphase FORTRAN programs are not supported unless they are cataloged via submit-to-batch and then loaded by specifying the name of the initial phase in the /LOAD job entry statement.

3. The Unit Assignment Table facility (described in *VS FORTRAN Installation and Customization*, SC26-3987) can be used by adding at the end of the FORTRAN source program

   ```
   /LOAD OBJECT
     INCLUDE IFYUATBL
   ```

4. The VS FORTRAN units 8 to 12 which correspond to the logical units SYS005 through SYS009 have a special meaning in the VSE/ICCF system that is delivered to you:

   - SYS005 corresponds to SYSIPT.
   - SYS006 corresponds to SYSLST.
   - SYS007 corresponds to SYSPCH.
   - SYS008 corresponds to SYSLOG for output.
   - SYS009 can be used to read from the punch area.

   These standards may be changed by tailoring VSE/ICCF (as described in *VSE/ICCF Installation and Operations Reference*) or via the /ASSGN job entry statement. The standards for units 8 through 99 may also be altered via the VS FORTRAN Unit Assignment Table IFYUATBL.

   In line with the above, you may use VS FORTRAN units 6, 9, or 12 to write to the terminal. To read from the terminal, you must use unit 5 or 8 with the /DATA INCON option.

5. The maximum number of source statements which can be processed by the VS FORTRAN compiler is a function of main storage availability. With the default value, approximately 500 source statements can be compiled. If all your comment statements are prior to the first FORTRAN statement, they will not be counted in the maximum number which may be processed by the compiler.

6. Subprograms called by a VS FORTRAN program may be in the same job stream in either source form or in object form. (See the example in section "Assembler Considerations" for how to set up a job stream with two separate programs which are to be loaded together). If a subprogram to be loaded with the main program is already in object form in the user's library, it may be included following the main FORTRAN source program using an /INCLUDE statement, that is:

```
/LOAD VFORTRAN
(FORTRAN Source Program)
/LOAD OBJECT (this statement is optional)
/INCLUDE SUBPGM1
/INCLUDE SUBPGM2
/DATA
 (input data if any)
```

The /LOAD OBJECT statement was optional. If an object deck is encountered in the job stream, the /LOAD OBJECT is implied.

Example:

The following example generates a list of the prime numbers from 2 to 1000 and displays this list on the user's terminal.

```
*READY
/input
/load vfortran
/file name=ijsys01,tracks=10
      (only required if no preallocated work files)
/file name=ijsys02,tracks=10
      (only required if no preallocated work files)
c     prime number generator
      write (3,1)
1     format ('1 following is a list of prime numbers'/
     *19x,'2'/19x,'3')
      do 4 i=5,1000,2
      k =sqrt (float(i))
      do 2 j=3,k,2
      if (mod(i,j) .eq. 0) go to 4
2     continue
      write (3,3) i
3     format (i20)
4     continue
      write (3,5)
5     format (' end of program')
      stop
      end
/endrun
*RUN REQUEST SCHEDULED
```

PL/I Considerations

VSE/ICCF supports the DOS PL/I Optimizing Compiler, and most PL/I functions are available to the VSE/ICCF user.

The PL/I Optimizing Compiler has many features that make it useful for both problem solving and data processing.

The PL/I compiler produces object programs which can be CALLed by, or may themselves call other programs written in PL/I or other languages (except BASIC and FORTRAN). See the *DOS PL/I Optimizing Compiler Programmer's Guide,* SC33-0008, for more information.

The PL/I Optimizing Compiler does not use the /OPTION job entry statement to obtain information concerning compile time options. Instead, it has its own control statement (called the PROCESS card) which, if used, should be the first record read by the compiler. The format of the PROCESS card is

```
* PROCESS   option1,option2,...optionn;
```

Refer to the Programmer's Guide for a complete list of the options and their meanings. When the compiler was installed, certain installation defaults were established. If these defaults match your requirements, you need not specify the * PROCESS card.

Some typical PROCESS cards are shown below:

```
* PROCESS       NOSOURCE,SYNTAX,NOCOMP,NODECK;  syntax check only
* PROCESS       NOSOURCE,DECK;  produce an object program
```

Notes and Restrictions:

1. Overlaid programs should not be built. A given program must consist of a single phase only to be processed by the LINKNGO utility. Programs loaded from a VSE library can contain more than one phase.

2. The implicit files SYSIN and SYSPRINT can be used for stream directed input/output without having to declare them as files. SYSIN will read data from the job stream while SYSPRINT will write data to the terminal.

3. If SYSIN is used for input, or if MEDIUM(SYSIPT,2540) or MEDIUM(SYS005,2540) are used for input, the request for input will become a conversational read to the terminal if the /DATA INCON facility is used. (Note that the normal card read programmer unit can not be SYS005 in your installation.)

4. The following are the preferred ways of defining the job stream input, terminal output and punch files:

   ```
   MEDIUM(SYSIPT,2540)
   MEDIUM(SYSLST,1403)
   MEDIUM(SYSPCH,2540)
   ```

   Even though the device type field for SYSIPT, SYSLST and SYSPCH is optional, it should be specified as above. SYS005, 6 and 7 (or equivalent installation defaults) can be used in place of SYSIPT, SYSLST and SYSPCH.

5. The PL/I compiler uses only work files IJSYS01(SYS001) and IJSYS02 (SYS002); if your system contains preallocated work files, the IJSYS00, 3 and 4 file areas may be used to pass data files from one job step to another even though a compilation may intervene.

6. The DISPLAY statement can be used to write data to the terminal and optionally to read data conversationally from the terminal. If the REPLY option of the DISPLAY statement is used for conversational reads, the amount of input data must not exceed 72 characters per read.

7. Any given program can use the temporary files specified by file names IJSYS00,1,2,3 and 4 (SYS000 through SYS004) for temporary data or for passing data between job steps. These file areas may be used for sequential record-oriented transmission or for stream-oriented transmission.

Example:

This example causes 80-character records to be read from the job stream and calculations to be performed. Both a stream-oriented file and a record-oriented file are built.

```
*READY
/input
/load pliopt
/include workfile
/option deck
* process link,compile,deck;
 myprog: proc options(main);
        declare ijsys03 file record output sequential
                  env(medium(sys003,3330) fb blksize(400) recsize(40)),
                    ijsys04 file stream output
                  env(f recsize(40) medium(sys004,3330)),
        1 recarea,
                  2 (a,b,c,x1,x2) float dec(6) complex;
        on endfile (sysin) goto endjob;
        open file (ijsys03), file (ijsys04), file (sysprint),
         file (sysin);
        loop:  get file (sysin) list (a,b,c);
               x1=(-b+sqrt(b**2-4*a*c)/2*a);
               x2=(-b-sqrt(b**2-4*a*c)/2*a);
               put skip file (sysprint) edit (recarea)(c(e(16,9)));
               put file (ijsys04) edit (recarea) (c(e(16,9)));
               write file (ijsys03) from (recarea);
               goto loop;
        endjob:  close file (ijsys03), file (ijsys04),
                     file (sysprint), file (sysin);
        end myprog;
/data
6 14 9
5 16 10
3 -12 2
/end
*READY
/run
*RUN REQUEST SCHEDULED
```

RPG II Considerations

RPG II is a powerful yet easy to use specification type language which can be used with VSE/ICCF. Together with the DSPLY function it enables you to create conversational programs.

The /OPTION statement is generally used to control compiler execution options; however, the RPG II control card (H in column 6) can be used to specify certain other options.

The RPG II language manual contains both language specifications and programmer information. There is no separate programmer's guide, but a user's guide for online RPG II operations is available.

An RPG II program can call or be called by other programs written in RPG II, PL/I, COBOL or Assembler as defined in the RPG II language manual. These other programs can be compiled or assembled in the same job stream or in a different job stream and stored in the VSE/ICCF library file or in a VSE library.

Notes and Restrictions:

1. The RPG II DSPLY statement can be used to write data to the terminal and to read data from the terminal. The file specification device field must be specified as CONSOLE for the DSPLY file.

2. Records read from the job stream can be read from a file set up with device type READ40 and Symbolic Unit SYSIPT or SYS005 (or equivalent installation default).

3. Lines printed on the terminal can be written on a file set up with device type PRINTER and Symbolic Unit SYSLST or SYS006 (or equivalent installation default).

4. Records to be placed in the punch area can be written on a file set up with device type READ40 and Symbolic Unit SYSPCH or SYS007 (or equivalent installation default).

5. To read records from the punch area, you can define a file as device READ40 with Symbolic Unit SYS008 (or equivalent installation default).

6. All unit record I/O (notes 2 through 5) should be specified as single buffered (no 2 in column 32 of the File Sheet) and fixed in length (column 19).

7. If you intend to use one of the temporary file areas, the file name on the File Sheet should be IJSYS0n where n is 0 through 4. The Symbolic Unit should be SYS00n. These file areas can be considered fixed or variable in format and can contain any block size permitted for the device type indicated. The device type must be specified as FBA, DISK14, DISK30 or DISK40. If preallocated work files are available within the system, these files need not be defined with /FILE statements.

8. The RPG II compiler uses IJSYS01 and IJSYS02 as work files. Thus when transferring data from one step to another past intervening RPG II compilations, do not use these file areas unless you are dynamically allocating these areas for each execution as DISP=KEEP files (on the /FILE statement).

9. The size of the relocation dictionary buffer for LINKNGO has been assigned a value of 1600. If you later find that very large RPG II programs are canceling or program checking, submit these jobs to batch for linkage by the VSE linkage editor programs.

Submit-to-Batch Capability

If your installation is using VSE/POWER, and your user profile allows you to use the SUBMIT facility, you can run jobs in batch VSE partitions. Your job streams can consist either of VSE/ICCF job entry statements, or of VSE job control language. The print output from the execution can be directed to the printer or held in a queue until you have viewed it. Also, if your installation is a VSE/POWER controlled node in a network, you can transmit jobs to be run at other nodes in the network.

Here are some reasons why you might want to submit to batch rather than to run a program in a VSE/ICCF interactive partition:

1. You want to use a program (for example, the VSE linkage editor LNKEDT) which you are unable to use in an interactive partition.

2. You want to use a feature which is not supported in your VSE/ICCF. (For example, multi-tasking or interval timer.)

3. The job you want to run is not urgent and you are not in a hurry for the results.

4. The job you want to run is very long and would tie up a VSE/ICCF interactive partition for too long.

5. The job you want to run produces a lot of print output which would take time to display.

6. Your 3270 display terminal has no printer but you need a hardcopy listing of some type.

7. You want to punch a member so it can be carried to a different location.

8. You want to link edit a multi-phase program using the linkage editor so that it can be run in an interactive partition.

There are undoubtedly many other reasons why you might submit a job for batch execution.

What Jobs May be Submitted

A library member consisting of either VSE JCL and VSE/POWER JECL or of VSE/ICCF job entry statements can be submitted for batch execution. Members containing VSE/ICCF job entry statements will have those statements automatically converted to VSE JCL while they are being submitted. Members consisting of VSE JCL can also contain the VSE/ICCF /INCLUDE statement to avoid the necessity of having all the data physically present within the job stream member.

A member that consists entirely of object statements (e.g. ESD, RLD, REP, END, TXT or SYM) is treated in the same way as a job containing VSE/ICCF job entry statements and can be submitted successfully. If the member also has control statements, e.g. ACTION, INCLUDE, PHASE statements, the member must be edited and a /LOAD OBJECT must be placed at the front of the deck to ensure proper execution after submission.

How to Submit a Job

The SUBMIT procedure is the basis for requesting the submit-to-batch function.

The SUBMIT procedure has two basic options:

1. The DIRECT option, which is the default. This option causes the job to be submitted for batch execution and the output of the job to be routed to the printer.

2. The RETURN option which causes all print and punch output to be held in the output queue. You can display the print output and then dispose of it as you see fit (see the /STATUSP and /LISTP commands).

   Only the submitter or the user designated in the DEST parameter of the VSE/POWER JECL statement * $$ LST may access an output queue entry. An authorized user may specify ANY in the DEST parameter (authorization is established in the VSE/ICCF user profile). This makes the output accessible to any user.

You can submit any type of VSE/POWER JECL with your job (however, the installation may have defined certain defaults which may cause some of your JECL parameters to be overridden). Your job stream may, in particular, contain a * $$ RDR statement. This allows you to include data from a diskette.

VSE/ICCF will allow more than one job to be run at a time using the submit-to-batch facility.

Submit a Job for Output on the Printer

To submit a job for execution in a VSE partition and to have the output written to the installation printer, enter:

```
SUBMIT name [DIRECT]
```

where the DIRECT operand is optional and the name operand is the name of the VSE/ICCF member which contains the job to be submitted. The job inherits the member name and will be queued for execution and run when the partition becomes available.

To test whether a job is waiting for execution, is executing or has completed, enter:

```
/STATUSP jobname
```

The /ERASEP command may be used to cancel the execution before it begins.

Submit and View Output at Terminal

To submit a job for execution in a VSE partition and to have the output stored in the VSE/POWER output queue so that you can view it later, enter:

```
SUBMIT name RETURN
```

where the name operand is the name of the VSE/ICCF member which contains the job to be submitted.

Use the /STATUSP command to see whether the job has been completed before you attempt to display it. Or wait until you are notified by VSE/POWER that the execution of your job finished.

Ask for the completion message with the /MSG command or set to automatic message display (/SET MSGAUTO ON).

To display the print output, enter the following command:

```
/LISTP jobname
```

You can use the /SKIP, /SHIFT, /LOCP, /COMPRES, /HARDCPY, /CONTINU and /CANCEL commands, and the GETP and GETL procedures, to control viewing and retrieval of your output. When you have finished viewing the output you should dispose of it in some way. To erase the output from the output queue, enter:

```
/ERASEP jobname
```

To route the print output to the printer, enter:

```
/ROUTEP LST jobname CLASS=class
```

where 'class' defines an output class for which a VSE/POWER printer task was started.

To route the print output to a remote VSE/POWER RJE printer, enter:

```
/ROUTEP LST jobname  REMOTE=remid
```

where remid is the numeric designation of the remote work station.

If you are using 3270 and you want to route the print output to a 328x hardcopy printer, enter:

```
/HARDCPY printer-name
 (clear key)
/LISTP jobname
 (clear key)
/CONTINU
 (wait for completion)
 (clear key)
/HARDCPY OFF
```

You can proceed with other work while the print output is being printed on the hardcopy device.

Note: *When you issue VSE/POWER interface commands such as /STATUSP, /LISTP, /ROUTEP, be aware that you receive messages from VSE/POWER. These messages are identified by the characters '1Q', '1R', and '1V' followed by a message code. For detailed explanations of these messages, consult the publication VSE/POWER Messages.*

Submit a Job for Transmission to Another Node

If your installation is a node of a VSE/POWER controlled network (PNET), and if you are allowed to supply your own VSE/POWER job entry control language (JECL) statements, you can submit jobs for transmission to another node. To do this you must prepare your JECL with the appropriate destination parameters (see *VSE/POWER Installation and Operations Guide*).

List Output Received from Another Node

When you expect output from jobs that were not submitted from one of your VSE/ICCF terminals, you should be aware of the following restrictions. This is particularly true if the job was submitted at a non-VSE system.

- List queue entries containing nondisplayable characters cannot be accessed by the /LISTP command or the GETL procedure. Nondisplayable characters are contained in the following types of print records:
  - SCS print records
  - BMS mapping records
  - 3270 format records
  - APA data records
- List queue entries with ASA or machine control characters are accepted up to a record length of 156 bytes. Longer records will be truncated.
- Only the submitter or the user designated in the DEST parameter of the * $$ LST statement is allowed to access a list queue entry (the VSE/ICCF administrator has access to all queue entries).
- Jobs submitted via the card reader are protected by an unprintable password if no password is provided. Only the VSE/ICCF administrator can retrieve output from such jobs.
- The jobnumber in the list queue will be different from the jobnumber of the job that created the list output.

Example:

User AAAA, logged on to VSE/ICCF at the node P2, wants to transmit a job for execution at node P1. Both P1 and P2 are VSE systems and nodes of a VSE/POWER controlled network (PNET). User AAAA is a VSE/ICCF administrator who may submit jobs, specify VSE/POWER JECL statements and use the /CP command.

1. The VSE/POWER JECL statements define where the job is to be run and the destination to which the output is to be routed:

```
/list tranp1_
 ...+....1....+....2....+....3....+....4....+....5....+ .CM
*READY
        _
         ...+....1....+....2....+....3....+....4....+....5....+.. .LS
        * $$ JOB JNM=TRANP1,CLASS=0,DISP=D,XDEST=P1,NTFY=YES
        * $$ LST CLASS=A,DISP=K,DEST=(P2,AAAA)
        * $$ PUN CLASS=A,DISP=K
        // JOB TRANP1
         . . .
        /&
        * $$ EOJ
```

   The XDEST parameter in the * $$ JOB statement defines the node where the job is to be run. The DEST parameter in the * $$ LST statement defines the node and the user to which the list output is to be routed. In this example, the DEST parameter could have been omitted (as was done for the * $$ PUN statement) because the originator of the job is the default destination.

2. User AAAA submits the job for execution to node P1:

```
submit tranp1 return_
 ...+....1....+....2....+....3....+....4....+....5....+ .CM
*READY
        _
         ...+....1....+....2....+....3....+....4....+....5....+..  .SP
        ***** START OF PROCEDURE (CLIST) *****
        K889I JOB TRANP1 00333 SUCCESSFULLY SUBMITTED ...
        *PARTIAL END PRINT
```

This is a normal submission of the job TRANP1. All routing information is contained in the VSE/POWER JECL statements. Job TRANP1 receives the jobnumber 00333 at node P2.

3. VSE/POWER notifies user AAAA when the job has been transmitted.

```
_
 ...+....1....+....2....+....3....+....4....+....5....+ .CM
*READY
        _
         ...+....1....+....2....+....3....+....4....+....5....+..  .CM
         02/15-14:39 MSG FROM VSE/POWER:
        1RA0I JOB TRANP1  00333 TRANSMITTED TO P1 FOR P1
        *END MSG
        *READY
```

User AAAA gets the message automatically (by pressing ENTER from time to time) if automatic message display is in effect (see /SET MSGAUTO... in *Chapter 3. System Commands, Procedures and Macros*).

4. User AAAA views the queue entry of the transmitted job in the reader queue of the target node, and transmits the appropriate command to the target node P1:

```
/cp pxmit p1,pdisplay rdr,cuser=aaaa_
 ...+....1....+....2....+....3....+....4....+....5....+ .CM
*READY
        _
         ...+....1....+....2....+....3....+....4....+....5....+....6....+..     .CM
         02/14-14:40 MSG FROM P1: 1R46I READER QUEUE   P D C S CARDS
         02/14-14:40 MSG FROM P1: 1R46I TRANP1    00157 3 D 0       10 FROM=P2(AAAA)
        *END MSG
        *READY
```

The target node responds with a series of messages, which could be mixed with other messages if AAAA is experiencing heavy message traffic. Job TRANP1 receives the jobnumber 00157 at node P1. It is awaiting execution.

5. The executing node notifies user AAAA when the job has been run. VSE/POWER at user AAAA's own node displays a message when the output was rerouted.

```
_
...+....1....+....2....+....3....+....4....+....5....+ .CM
*READY
```

```
_
...+....1....+....2....+....3....+....4....+....5....+....6... .CM
 02/15-14:52 MSG FROM P1:
1Q5DI  EXECUTION COMPLETED FOR TRANP1   00157 ON P1, TIME=14:52:48
 02/15-14:54 MSG FROM VSE/POWER:
1RB5I OUTPUT TRANP1  00334(00333) RECEIVED FROM P1  FOR P2
*END MSG
*READY
```

The output is given the new jobnumber 00334 at node P2. The jobnumber of the job is given in brackets (00333). The list output can now be displayed in the usual way.

Restrictions and Notes:

- The SUBMIT program accepts only * or // as start of a VSE job. Anything else is considered VSE/ICCF JCL and handled accordingly. Place all of your ASSGNs immediately after the JOB statement, or use the PERM option of // ASSGN.
- SUBMIT transforms /DATA into /*
- If the first data record after VSE/ICCF job entry statements starts with // followed by a blank, SUBMIT moves this record in front of the // EXEC statement of the generated job stream.
- /LOAD DTSDUMMY in a job stream to be submitted is **not** transferred into a corresponding // EXEC DTSDUMMY statement.
- The SUBMIT program translates VSE/ICCF JCL into VSE JCL when possible and necessary. /OPTION GETVIS=nnK is translated to // EXEC program,SIZE=P - nnK (where P is the default size of an interactive partition). If nnK is greater than P, // EXEC program, SIZE=AUTO will be generated. /OPTION GETVIS=P - nnK causes // EXEC program,SIZE=nnK to be generated.
- Do not use the following names with VSE/POWER as library member names: ALL, HOLD, FREE, RJE, LOCAL. Also, do not use names consisting of only one character; when submitted to VSE/POWER as jobnames, they may conflict with VSE/POWER operator commands.
- Columns 73-80 of VSE/POWER JECL records are not transferred by SUBMIT.
- Because VSE/POWER transmits only 60 characters per message, its diagnostic messages may be truncated. This may happen for all commands listed in *Job Execution under VSE/POWER* in Chapter 1.
- If the RETURN option is to be used, it is recommended that RBS should equal 0 in the LST and PUN cards.
- Comments on VSE/POWER JECL statements are not submitted to VSE/POWER.

- Trying to assign a new class to a job by placing a '* $$ CTL' statement in the job stream will have no effect in a job that was submitted via the SUBMIT procedure. SUBMIT always specifies the CLASS parameter in the '* $$ JOB' statement. Therefore, the '* $$ CTL' statement is ignored.

- If a continuation character is encountered in column 72 of a VSE/POWER JECL statement and no correct continuation statement is following, the continuation character is ignored.

Appendix A. Interactive Job Restrictions

The following restrictions apply to programs that are to be run in interactive partitions.

Access Methods and File Definitions

- The trackhold option (HOLD = YES) must not be used with any of the access methods SAM, DAM, and ISAM.

- BTAM-ES as well as ACF/VTAM cannot run in interactive partitions, nor can a program in an interactive partition link to ACF/VTAM.

- Access methods for MICR/OCR devices must not be used by programs in interactive partitions.

- The support of system files (SYSFIL = YES) does not affect programs in interactive partitions because VSE/ICCF directs the I/O to these logical units, to the terminal, or to the VSE/ICCF library file.

- Except for the unit record devices for which I/O is intercepted by VSE/ICCF (card reader assigned to SYSIPT and usually to SYS005; printer assigned to SYSLST and usually to SYS006; punch assigned to SYSPCH and usually to SYS007, punch input file usually assigned to SYS008 and console assigned to SYSLOG and usually to SYS009), assignments to physical units have to be done at or before VSE/ICCF startup. The assignments are valid as long as VSE/ICCF is up.

  This restriction may cause problems for programs that use access methods other than those for DASDs. In DASD access methods, OPEN automatically replaces the logical unit supplied in the program with the one supplied in the label information. If such a program is run in an interactive partition, the assignment required has to be set up at or before VSE/ICCF startup time.

  VSE provides only one set of logical units (SYSnnn) for all interactive partitions, whereas each VSE partition has its own set of logical units. That means, if a program to be run in an interactive partition needs a dedicated logical unit assigned to a file, no other program may run in an interactive partition which needs the same logical unit assigned to a different file.

  If assignments have not been made, VSE/ICCF sets up dummy assignments for all unit record devices at startup time. I/O to these devices is intercepted by VSE/ICCF; that is, the system logical units and the corresponding default logical units which are defined as VSE/ICCF tailoring options (usually SYS005, SYS006, SYS007, SYS008, SYS009).

  A default logical unit can only be used in a program for a different file if the default assignment has been changed via the /ASSGN job entry statement.

- VSAM space management for SAM files can be used in interactive partitions by specifying TYPE = VSAM on the /FILE statement. These files should only be obtained via the /FILE statement, and not at VSE/ICCF startup time. The RECORDS and RECSIZE parameters of the

DLBL JCL statement are not supported on the /FILE statement. It is your responsibility to ensure that your file identifications are unique, for example by specifying the IDENT parameter with the special parameters &USR, &TRM and &PRT. VSE/ICCF dynamic file management is recommended for obtaining workfile space.

- For conversational reads from terminals, double buffering with overlap of I/O cannot be used; otherwise input and output messages could get out of sequence at the terminal. Blocking of records is ignored by VSE/ICCF for all those unit record devices for which the I/O is intercepted by VSE/ICCF. VSE/ICCF always assumes that a block contains only one record. Moreover, only fixed length records are supported for these devices. VSE/ICCF assumes that the first byte of the record represents the first byte of data.

- The maximum number of extents for a CKD-DAM file which can be opened by a program in an interactive partition is 97.

- The label information which VSE/ICCF builds from the /FILE job entry statements is not written to the label area on SYSRES but to the beginning of the interactive partition. OPEN reads the file label information from this area. The maximum number of different files which can be specified via /FILE statements during one job is 28.

- VSE/ICCF does not provide any more protection against simultaneous access of files than VSE. Unpredictable results may thus occur if several interactive partitions access the same file at the same time. VSE/ICCF, however, provides a facility which allows each interactive partition to have its own set of work files.

- If jobs are to be run a number of times in an interactive partition, ASSGN and EXTENT statements should be specified in addition to the DLBL statement, for example for VSAM files. This prevents the partition from running out of LUBs, which are not freed at EOJ. Another safeguard against this situation is to specify the maximum number of LUBs using the job control command NPGR.

- For VSAM files the default value of GETVIS space is not sufficient. Define more with the job entry statement /OPTION GETVIS=....

Librarian and Linkage Editor

- The VSE Librarian program provides service functions for VSE libraries. The three VSE/ICCF macros: LIBRC, LIBRL, LIBRP make the access to VSE libraries transparent to the terminal user. These macros allow the terminal user to catalog members into a VSE library, to list a member of a VSE library, and to transfer a member from a VSE library into a VSE/ICCF library. Because the macros cause the VSE Librarian program to be run, the restrictions pertaining to the VSE Librarian program hold for those macros as well.

- The VSE linkage editor cannot run in an interactive partition.

- LSERV can only run in **one** interactive partition at a time.

Sort/Merge

The STORAGE=(n, VIRT) parameter must be specified in the SORT MERGE OPTION control statement because page fixing (SVC 67) is not supported in interactive partitions.

Note: You must supply labels for SORTWK1 and SORTWK2 if necessary. VSE/ICCF changes the filenames to IKSYSP3 and IKSYSP4 before OPEN depending on the /OPTION PERMFILE setting.

Special Considerations for Interactive Partitions

- From the 15 Storage Protect keys available to VSE, VSE/ICCF has 15 (minus the number of generated VSE partitions) for protecting its interactive partitions. Assuming that the VSE system has been generated with 5 VSE partitions, VSE/ICCF would then be able to use up to 10 Storage Protect keys for its interactive partitions. If VSE/ICCF had been tailored with 20 interactive partitions, then 2 interactive partitions always get the same Storage Protect key. VSE/ICCF makes sure that interactive partitions with the same Storage Protect key are not located adjacent to each other.

  If two or more interactive partitions have the same Storage Protect key, a program in one interactive partition can accidentally damage a program in another interactive partition with the same Storage Protect key.

- Programs in interactive partitions are run under control of VSE subtasks. Because there are not enough VSE subtasks available for all the programs that can be run concurrently in interactive partitions, VSE/ICCF performs rollin/rollout of programs on a time-slice basis. Rollout means that a subtask is taken away from a program, whereas rollin means that a subtask is assigned to a program. The dispatching priorities are randomly assigned to the programs and usually change during the execution of a program.

  If sufficient VSE subtasks are available for interactive partition execution, VSE/ICCF does not do rollin/rollout but leaves the subtasks assigned until the programs terminate. If in such an environment a program with very little I/O activity gets a high dispatching priority, it will monopolize the total background execution of VSE/ICCF.

- The layout of interactive partitions is very similar to the layout of VSE batch partitions, except that up to 4K of space at the beginning of each interactive partition is reserved for VSE/ICCF.

  It is not possible to define an amount of real storage for an interactive partition which can be fixed permanently (PFIX/PFREE).

- The communication region of an interactive partition is located at an address higher than 32K and is pageable. A program gets the address of its communication region only by issuing the COMRG macro.

Supervisor Services

- The JOBCOM macro is not supported in interactive partitions.

- The PDUMP macro causes up to 8K (or until print area full condition) to be printed to the terminal. Everything beyond this point is ignored.

- The DUMP and JDUMP macros cause the VSE/ICCF DUMP program to be invoked if it had been loaded into the SVA before VSE/ICCF was started. If the VSE/ICCF DUMP program had not been loaded into the SVA, the program is canceled without any dump. If /OPTION DUMP was specified, the work area for the VSE/ICCF DUMP program is taken from the GETVIS area of the interactive partition, or the last 3K of the interactive partition are used as work area.

- All VSE messages, except information (I) type messages supplied from SVA or B-transient routines, are routed to the terminal user requesting the service and to the console operator. Answers to such messages are only read from the operator console.

- The ASSIGN macro is supported in interactive partitions. Note, however, that temporary assignments are not automatically reset at end of a job as they are in a VSE batch partition. You are responsible for resetting these assignments. If temporary assignments are not reset, the pool of logical units could eventually become exhausted. This would prevent new assignments from being made until VSE/ICCF had been shut down and brought up again.

- If a B-transient service that produces printer output fills the requestor's print area, any additional output from the same B-transient service is ignored.

- Checkpoint/restart is not supported for interactive partitions.

- The following table shows for each SVC supported in VSE whether or not it may be used by a program in an interactive partition. If an SVC indicated as 'not supported' is used by a program in an interactive partition, the program will be canceled. However, if the program issues an SVC which is indicated as being 'ignored' the program will not be canceled but the SVC function is skipped. 'Supported' means that the same rules apply regardless of whether the program that issued the SVC is running in an interactive partition or in a VSE batch partition.

SVCs Supported in Interactive Partitions

SVC	Macro	Supported	Not Supported	Ignored
0	EXCP	x		
1	FETCH	x		
2		x		
3		x		
4	LOAD	x		
5	MVCOM	x		
6	CANCEL	x		
7	WAIT	x		
8		x		
9	LBRET	x		
10	SETIME	x		
11		x		
12		x		
13		x		
14	EOJ	x		
15			x	
16	STXIT (PC)	x		
17	EXIT (PC)	x		
18	STXIT (IT)	x		
19	EXIT (IT)	x		
20	STXIT (OC)	x		
21	EXIT (OC)	x		
22			x	
23			x	
24	SETIME	x		
25			x	
26		x		
27			x	
28	EXIT (MR)		x	
29	WAITM		x	
30			x	
31			x	
32	——			
33	COMRG	x		
34	GETIME	x		
35			x	
36	FREE		x	
37	STXIT (AB)	x		
38	ATTACH		x	
39	DETACH		x	
40	POST	x		
41	DEQ		x	
42	ENQ		x	
43	——			
44		x		
45			x	

Figure A-1 (Part 1 of 3). SVCs Supported in Interactive Partitions

SVC	Macro	Supported	Not Supported	Ignored
46			x	
47	WAITF		x	
48			x	
49			x	
50			x	
51		x		
52	TTIMER	x		
53			x	
54			x	
55			x	
56		x		
57		x		
58			x	
59			x	
60		x		
61	GETVIS	x		
62	FREEVIS	x		
63		x		
64		x		
65	CDLOAD	x		
66	RUNMODE	x		
67	PFIX		x	
68	PFREE		x	
69	REALAD		x	
70	VIRTAD		x	
71	SETPFA		x	
72			x	
73			x	
74			x	
75	SECTVAL	x		
76			x	
77		x		
78	CHAP	x		
79			x	
80	SETT		x	
81	TESTT		x	
82	(special ICCF SVC)	x		
83			x	
84			x	
85	RELPAG			x
86	FCEPGOUT		x	
87	PAGEIN		x	
88	TPIN		x	
89	TPOUT		x	
90			x	

Figure A-1 (Part 2 of 3). SVCs Supported in Interactive Partitions

SVC	Macro	Supported	Not Supported	Ignored
91			x	
92	XECBTAB	x		
93	XPOST	x		
94	XWAIT	x		
95	EXIT (AB)		x	
96	EXIT (TT)		x	
97	STXIT (TT)		x	
98	EXTRACT	x		
99		x		
100			x	
101		x		
102			x	
103		x		
104		x		
105		x		
106		x		
107			x	
108		x		
109		x		
110	LOCK/UNLOCK	x		
112		x		
113	XPCC	x		

Figure A-1 (Part 3 of 3). SVCs Supported in Interactive Partitions

Notes:

1. *CANCEL ALL is changed by VSE/ICCF to CANCEL.*
2. *Fast CCW-translation is skipped for interactive partitions.*
3. *EXCP with the parameter REAL is not supported in interactive partitions.*
4. *Programs which use VSE services or access supervisor tables which are not described as official interfaces in VSE/Advanced Functions Application Programming: Macro Reference, SC33-6197, may cause problems if they are run in interactive partitions, although they run without problems in VSE batch partitions.*
5. *Applications of XPCC that make the use of WAITM necessary cannot be used (e.g. multiple connections).*

Analysis of CCWs for Unit Record I/O

I/O requests to the card reader, the printer, the punch, the punch input file, and the operator console are intercepted by VSE/ICCF. These requests are routed to either the terminal or to the VSE/ICCF library file. To do this, VSE/ICCF analyzes and transforms the CCW of each I/O request.

To avoid a complex CCW analysis routine, which would affect system performance, VSE/ICCF puts restrictions on the CCW chains that it intercepts. Usually these restrictions do not cause problems. However, if your program output looks unusual, you might have violated one of these restrictions. The following list explains these restrictions:

- Analysis of CCW Chains:

  VSE/ICCF generally assumes that each CCW in a chain is adjacent to the former one; that is, VSE/ICCF expects the next CCW to be located at an address which is 8 bytes higher than that of the last CCW. Status modifier commands are ignored with respect to the flow of control through the CCW chain.

  Depending on the chaining flags, VSE/ICCF continues analyzing the CCW chain. A TIC command always causes VSE/ICCF to proceed with the next CCW at an address contained in the TIC command. CCWs that are chained together via the data chaining flag are handled by VSE/ICCF as if they were CCWs with the same command code chained together via the command chaining flag.

  The command code checking is very restricted. For non-DASD channel programs, the analysis is as follows: If the read flag (bit 6) is on in the command code, VSE/ICCF handles the CCW as read CCW. If the write flag (bit 7) is on, VSE/ICCF assumes that the CCW is a write CCW. And if the control flags (bits 6 and 7) are on, VSE/ICCF handles the CCW as control CCW. The command code X'08' is handled as TIC CCW. VSE/ICCF ignores the modifier bits of the command code.

  Command code checking for DASDs (channel programs beginning with opcode X'07') allows only the command codes X'05', X'0D' and X'ID' to be treated as writes and command codes X'06', X'0E' and X'IE' to be treated as reads. All other command codes, except the TIC (X'08'), are ignored. The TIC is handled as described above.

  VSE/ICCF ignores commands that do not fall into the above categories. It also ignores commands that are not meaningful for the logical unit; for example, read from SYSLST. VSE/ICCF does not issue a message nor is the I/O request canceled. If such a CCW has a chaining flag on, VSE/ICCF proceeds with the next CCW following the invalid one.

- Write Requests to SYSLST or the Corresponding SYSnnn:

  The maximum number of bytes you can write to your print area with one write CCW is 156. If the data length is longer, the remainder is ignored.

  If you use DTFDI with RECSIZE not equal to 121, VSE/ICCF does not strip off the first character (which is supposed to be the printer control character). This character therefore appears in your print area as data. If you issue your own EXCP requests to the printer, you have to be careful with data lengths of 121, 129, and 133. If the data length is 129, VSE/ICCF strips off the first 9 bytes, and if the data length is 121 or 133, VSE/ICCF strips off the first byte.

  VSE/ICCF provides only one byte per write CCW for saving printer control information. This means that the control information of only one command can be saved per write request. Other control information gets lost. The VSE/ICCF RELIST utility uses the control information to build the appropriate CCW chain for the central printer. If the command code of a write CCW is 'write without spacing', VSE/ICCF saves (for the RELIST utility) the command code of the most recent control command it encountered in the CCW chain. If the write CCW is not a 'write without spacing', VSE/ICCF usually saves the command code of the write CCW. However, if the most recent control command is a 'skip to channel 1' request, VSE/ICCF saves a special code which allows RELIST to generate the 'skip to channel 1' request in addition to the original write request. For the following write CCWs, VSE/ICCF assumes the most recent control command to be 'space 1 line' until it encounters another control command. VSE/ICCF does no error checking for valid printer control commands during CCW analysis.

VSE/ICCF provides special code to support DASD channel programs and tape channel programs on SYSLST or the corresponding SYSnnn. However, VSE/ICCF does not transfer printer control information to the RELIST utility in a meaningful manner.

The device type codes of the 3800 Printing Subsystem should not be specified because its I/O module buffers the printer output and displays it only when the buffer is full or at close time.

- Read/Write Requests to/from SYSLOG or the Corresponding SYSnnn:

  Write requests to SYSLOG (or the corresponding SYSnnn) go to your print area in the same way as they go to SYSLST. Therefore, write requests to SYSLOG or the corresponding SYSnnn are handled in the same way as those to SYSLST.

  Independent from the data length you specify in a read CCW, VSE/ICCF always accepts up to 256 characters as input from the terminal. If the input data length from the terminal is longer than that specified in the read CCW, VSE/ICCF stores the difference as residual count to the corresponding CCB and moves the data up to the length specified in the read CCW to the I/O area of your program. If the input data length from the terminal is smaller than that specified in the read CCW, VSE/ICCF posts the residual count to 0 and moves the data up to the length of the input from the terminal to the I/O area of your program. The remainder of the I/O area remains unchanged.

  It is meaningful to specify a data length of only up to 256 in the read CCW.

- Read Requests from SYSIPT or the Corresponding SYSnnn:

  If you specified /DATA INCON for your program, which means that a read request from SYSIPT or the corresponding SYSnnn is directed to the terminal, the same rules apply as described for read requests from SYSLOG, except that the data moved to the I/O area of your program can never be larger than 80 bytes. If the input data length from the terminal is larger than 80 bytes, the residual count in the corresponding CCB is set to 1. If the input data length is smaller than 80, the residual count is set to 0.

- Print and Punch Area, and 'SYSPCH to a Member':

  The print and punch areas are allocated by VSE/ICCF on request in multiples of 100 records. Before anything is put into these records, VSE/ICCF initializes them with /* */ characters. These are overridden one after the other by the data you write to the area. If your output needs a number of records not equal to (n*100 - 1), the remainder of the area still contains the initialization characters. (Note that data to SYSLST or the corresponding SYSnnn with a data length larger than 78 requires 2 records from the print area).

  If you punch output to an existing member, the data first overrides the existing member. If all records of the existing member have been used up, then the same allocation and initialization mechanism takes place as for the punch area. This means that if your punch output consists of more records than the existing member, then you may find /* */ records at the end of the punch output. However, if your punch output consists of fewer records than the existing member, you will find that the old information (from the existing member you punched to) has been deleted.

Appendix B. Context Editor

The main difference between the context editor and the full screen editor (see Chapter 4. Editor) is the following. Full screen editor commands can be entered in the command area and in the line command areas. Data may be entered anywhere on the screen.

Context editor commands, on the other hand, can be entered only in the command input line.

In general, the command sets of the two editors are identical. However, there are exceptions: some commands are valid only under the full screen editor, others are "useful" only under the context editor.

The context editor must be used if

- You work with a 274x terminal.
- If you write procedures and macros that perform editing functions.

Note: As with the full screen editor, all context editing commands take effect immediately.

Invoking the Context Editor

You invoke the context editor by issuing the /EDIT command which places your terminal into context editor ('ED') mode. To edit a library member, specify the member name, followed by the member password if required:

```
/EDIT membername [passwd]
```

To edit the input area, enter the /EDIT command without member name.

Editing Modes

As with the full screen editor, you are editing in two basic modes:

- edit mode
- editor input mode

After invoking the context editor, your terminal is in context editor mode. You can enter commands to move the line pointer and to locate, insert, add, or replace data. The INPUT command transfers your terminal into editor input mode. Editor input mode is indicated by the message MORE INPUT?

In editor input mode, you may enter only data. Each line that you enter in editor input mode is added to the file at the position of the line pointer. One 80-character line is created from each input line. To insert a blank line in a file, type at least one space and press carriage return or ENTER.

You terminate editor input mode when entering a null line. The message INPUT MODE ENDED indicates that you are back in edit mode.

If a null line is entered in edit mode, it has the same effect as the NEXT command.

Response Modes

There are three editing response modes (or verification modes), which are controlled by the VERIFY command:

- verify mode
- brief mode
- long verify mode

Verify is the normal mode. It automatically displays each line that has been changed or found as a result of a command.

Brief mode does not display the specified lines, so that you must issue a PRINT command to obtain a display.

The long verify mode only pertains to 3270 terminals and is most useful in a local 3270 environment. In long verify mode, the output area of the screen contains a number of lines before and after the line pointer. A scale line indicates the location of the line pointer. The number of lines before and after the scale line can be varied via operands to the VERIFY command. Long verify mode is automatically set for local 3270 terminals. (If the long verify display is lost, it can be restored by entering VERIFY with no operands.)

Certain messages are always printed even if you have selected brief message mode.

Summary of Editor Commands Valid in Context Editing

The table below lists the editor commands that are valid under the context editor. Most of these commands are described in "Chapter 4. Editor". Others are equivalents of system commands with the same name, for example: LIBRARY as equivalent of the system command /LIBRARY. In these cases, the descriptions in "Chapter 3. System Commands, Procedures and Macros" apply.

The editor commands basically work the same way under both editors. However, there are some differences. Where such difference exists, the pertinent command in the table is flagged by an '*', and the following sections will present some explanations.

COMMAND	FUNCTION
Add	Adds data beyond the end of data within the zone.
ALIgn	Aligns data in the zone to right and left margins.
ALter	Changes a single character to another character in one or more lines.
BAckward*	Moves the line pointer upwards a given number of lines; see UP command.
BLank	Blanks out characters in the current line.
Bottom	Moves the line pointer past end-of-file.
BRief*	See the VERIFY command.
CAse	Controls upper/lower case input data translation.
CENter	Centers the data within the current zone.
Change*	Changes one character string to another in one or more lines.
CTL	Sets various VSE/ICCF options; works like the /SET system command.
DELete	Deletes one or more lines.
DELIM	Defines the delimiter character.
DOwn	Moves the line pointer forward a given number of lines; see the NEXT command.
DUP	Duplicates the current line a specified number of times.
ECHO	Writes a message to the terminal; works like the /ECHO system command.
END	Terminates edit mode; see the QUIT command.
FILe*	Saves the file and terminates the edit session.
Find	Performs a column dependent search for specified data.
FLag	Sets change flagging on or off.
FOrward*	Moves the line pointer forward a given number of lines.
GETfile	Copies data from another member or work area into the file that you are editing.
HARdcpy	Places the terminal in hardcopy mode and directs terminal output to a hardcopy printer or to a private destination; works like the /HARDCPY system command.
IMage*	Sets backspace/tab transparency on or off.
INDex	Builds an index of the current file for rapid editing.
INPut*	Sets the editor to editor input mode for entry of multiple lines of data.
Insert*	Inserts a single line of data into the file.
JUStify	Left or right justifies data within the zone.
LIBRARY	Displays the member names of a library; works like the /LIBRARY system command.
LINemode*	Sets linenumber editing on or off.

Figure B-1 (Part 1 of 3). Editor Commands in Context Editing

COMMANDS	FUNCTION
LN	See LOCATE command.
Locate	Locates a string of characters within a file.
LOCNot	Locates the first nonoccurrence of a string; see LOCATE command.
LOCUp	See LOCATE command.
LUp	See LOCATE command.
MSG	Displays messages that have been sent to the terminal; works like the /MSG system command.
Next	Moves the line pointer forward a given number of lines.
Overlay	Overlays the current line with the characters specified in the operand field of the command.
OVERLAYX	Overlays the current line with hexadecimal format data.
OX	See OVERLAYX command.
PF	See PRINT command.
PFnn	Invokes the function associated with PF key 'nn'; works like the /PFnn system command.
POint	Positions the line pointer based on an established index.
Print	Displays a given number of lines.
PRINTFwd	See PRINT command.
PROmpt	Sets or displays prompt increment for line number editing.
Quit	Terminates edit mode.
RENum	Puts new sequence numbers into the file being edited.
REPEAT	Executes a subsequent OVERLAY, BLANK, ALIGN, CENTER, JUSTIFY or SHIFT command a given number of times.
REPlace*	Replaces a library member with all or part of the input area, or with a newly created file; works like the /REPLACE system command.
REStore	Restores various editor settings from previous STACK EDIT.
Rewrite*	Replaces the current line with new data.
RPT	See REPEAT command.
SAve*	Saves the contents of the input area or of a newly created file; works like the /SAVE system command.
Search	Searches the file from the top for a given string of characters.
SET*	Sets various VSE/ICCF options; works like the /SET system command.
SHIft	Shifts data within the zone left or right a given number of columns.
SHow*	Displays the setting of all VSE/ICCF options that can be controlled by the SET command; works like the /SHOW system command.
SPlit	Splits the current line into two lines.
STACk	Places data or commands in the editor stack.
STATUS	Displays the setting of all VSE/ICCF options that can be controlled by the SET command; works like the /SHOW system command.

Figure B-1 (Part 2 of 3). Editor Commands in Context Editing

COMMANDS	FUNCTION
TABset	Establishes logical tab settings; works like the /TABSET system command.
Top	Positions the line pointer to the null line in front of the first line in the file.
TYpe	Displays a given number of lines; same as the PRINT command.
Up	Moves the line pointer upwards a given number of lines.
Verify*	Sets verifications on or off; or sets 3270 full screen verify; or displays the current line.
Zone	Sets the current zone for editing, or scanning operations.
'nn'	Replaces or locates lines by sequence number.

Figure B-1 (Part 3 of 3). Editor Commands in Context Editing

Peculiarities in Context Editing

The editor commands flagged in the above table by an asterisk either work differently under the context editor (as opposed to the full screen editor), or they are not relevant in full screen editing. The following sections explain the peculiarities.

BACKWARD Command

The BACKWARD command in context editing works the same way as the UP command. That means, you can move the line pointer upwards a given number of lines. The BACKWARD command does not allow you to scroll upwards a specified number of logical pages (as in full screen editing).

BRIEF Command

Refer to the description of the VERIFY command below.

CHANGE Command

If you enter this command with no operands, instead of moving the cursor to the current line of the screen, the record pointed to by the line pointer appears in the input line of the display. After that, the system offers you several facilities:

- You can alter the contents of the input line by retyping part or all of the line, or by using the INSERT, DELETE, or ERASE-EOF keys to insert or delete characters.
- When the line is modified, press ENTER, which will cause the record in the input line to replace the current line in the output display area.
- If the column suffix is used, the cursor is placed at the location indicated by the column suffix.
- If you do not want to change the line, press the ERASE INPUT key and then the ENTER key and the line will not be changed.

Note: A CHANGE command without operands is invalid at a typewriter terminal.

FILE Command

Under the context editor, the FILE command allows saving **part** of the input area as a library member. This is not possible in full screen editing. For the syntax of the FILE command refer to the /SAVE system command.

FORWARD Command

The FORWARD command in context editing works the same way as the NEXT command. That means, you can move the line pointer forward a given number of lines. The FORWARD command does not allow you to scroll forward a specified number of logical pages (as in full screen editing).

IMAGE Command

The IMAGE command is used to control how the context editor handles backspace and tab characters within commands or input data.

```
IMage          [ON|OFF]
```

ON causes backspaces or tabs to be removed according to their logical functions. Backspaces cause the previous character to be logically overlaid. Tabs cause blanks to be copied according to predefined logical tab stops.

OFF causes tab and backspace characters to be treated like other characters.

If no operand is specified, the current IMAGE setting is displayed. The IMAGE command only applies to the current editing session.

Example:

The example assumes that '#' is defined as the backspace character.

```
     request:        image on
                     insert xxx###zzz
                     print

     response:       zzz

     request:        image off
                     insert xxx###zzz
                     print

     response:       xxx###zzz
```

Note: The IMAGE command is also available in full screen editing. But it has no practical use there.

INPUT Command

In context editing the INPUT command has the same effect as in full screen editing: the terminal changes from edit mode to editor input mode.

However there are some differences:

- When linenumber editing is in effect, sequence numbers can be made part of all new lines added to the file (see the PROMPT command in "Chapter 4. Editor"). In full screen editing, the sequence numbers may be overwritten by data.
- If in context editing, you leave input mode without having entered a line, the line pointer remains at the position that it was in just before editor input mode was entered. In full screen editing, the line pointer is advanced by one line.
- When using the 3270 you can press the Cancel (PA2) key[1] to return from editor input mode to edit mode. If the PA2 key is pressed a second time, it has the same effect as the QUIT command. Edit mode is terminated and command mode is entered.

INSERT Command

As under the full screen editor, the INSERT command is used to insert a string of characters into the file without entering editor input mode.

In context editing the INSERT command without operands is invalid; the message INVALID OPERAND appears. On the other hand, in full screen editing, the INSERT command has the same effect as the INPUT command.

1 The definition of the Cancel key is a VSE/ICCF tailoring option. The default is PA2. It may be different for your system.

Note: If you are using linenumber editing, the insert string will receive a sequence number; under the full screen editor this does not happen.

LINEMODE Command

Regarding the LINEMODE command, the environment in context editing is slightly different from that in full screen editing.

These are the differences:

- When you use linenumber editing under the context editor, you are prompted with a line number to enter each line.
- While in full screen editor input mode, the columns occupied by the sequence number should not be used as part of the input lines. Otherwise the sequence number will be overwritten.

REPLACE Command

In context editing the REPLACE command works the same way as the /REPLACE system command. Therefore you can replace an existing library member with only a portion of the input area. If only a portion of the input area replaces the existing library member, the portion not participating in the replace will remain in the input area.

For the syntax of the REPLACE command refer to the /REPLACE system command.

REWRITE Command

The REWRITE command is used to replace the current line with 'string'.

```
Rewrite       string|/string/
```

string	is an input line that replaces the current line.
/	delimiter character

The column suffix can be used with this command.

The logical tab character, the tab key, and the logical backspace character (subject to the IMAGE setting) can be used in 'string'. 'String' is separated from the command by only one blank unless the delimiter characters are used.

The line pointer is not advanced by this request.

The REWRITE command is valid when linenumber editing is in effect as long as the sequence number field begins in column 1, 73 or beyond.

The keyboard is unlocked. For 3270 terminals, the replacement line is displayed.

Examples:

```
1.  Rb¬IREG = J + K**2  (where ¬ is the tab character)
    line before request:           IRE555 = 1 - K
    request:                 r ¬ireg = j + k**2
    line after request:            IREG = J + K**2

    The 'string' specified with the request replaces the current
    line.  Assuming that the logical tabs are set for a FORTRAN
    filetype, the statement IREG = J + K**2 begins in column 7.

2.  RC16 THIS IS THE NEW LINE
    line before request:     THIS IS THE ORIGINAL LINE
    request:                 rc16 this is the new line
    line after request:                     THIS IS THE NEW LINE

    The 'string' specified becomes the new line but it does not
    start until column 16.  Columns 1 through 15 are replaced with
    blanks.
```

Note: The REPLACE command is also available in full screen editing, but it has no practical use there.

SAVE Command

While the terminal is in context edit mode, the SAVE command has the same effect as the /SAVE system command.

The differences between the two editors are as follows:

- In context editing you are able to save only a part of the contents of the input area. All lines not saved remain in the input area, and all lines saved are no longer in the input area at the end of the SAVE operation.
- Under the context editor, the 'name' parameter must always be specified.

For further explanations see the /SAVE system command.

SET Command

The SET command offers the same functions under both editors. However, the following options are not available in context editing:

- NULls
- NUMbers
- PFC
- REPoption

SHOW Command

In context editing the SHOW command does not accept the 'NAMES' operand. This is only possible in full screen editing.

VERIFY Command

The VERIFY command controls what is displayed while you are entering editor commands that modify lines within the file.

The BRIEF command has no effect when used during full screen editing.

```
Verify       OFF
BRief

Verify       [ON] [n1|72]
             LONG [n1|72 [n2|20[n3|7]]]
             FULL [n4|1]
```

n1 is a decimal number from 1 to 80 indicating the number of columns of data to be displayed when verification mode (the opposite of brief mode) is in effect. The default is the current line size value at the time the editor is entered, which is usually 72.

n2 is a decimal number indicating the total number of lines to be displayed on the screen when LONG verify mode is set. The values for the various screens are:

```
n2=5-23 for 3278 model 5 (default 23)
n2=5-28 for 3278 model 3 (default 28)
n2=5-39 for 3278 model 4 (default 39)
n2=5-20 for all other models (default 20)
```

n3 is a decimal number from 1 to n2 indicating the number of lines (of the total displayed in LONG verify mode) which are to be displayed before the current line. (The scale line which precedes the current line is included in this value.)

n4 is a decimal number from 1 to 16 indicating the number of lines prior to and including the current line to be displayed on the full screen edit or logical screen.

There are three basic verification modes:

BRIEF MODE	Changed lines are not displayed.
VERIFY MODE	Changed lines are displayed.
LONG VERIFY	(3270 only) Several lines before and after the current line are displayed.

Verify mode is the default for remote 3270 terminals and for hardcopy terminals. Long verify mode is the default for local 3270 terminals.

If one of the following is specified:

```
VERIFY
VERIFY nn
VERIFY ON nn
```

verify mode will be set if brief mode was in effect. If verify mode or long verify mode was in effect, that mode will be retained. The verify or long verify information will be written to the terminal. If nn is specified, the width of the verification display will be set to the number of columns indicated. If nn is not specified, the number of columns displayed is determined by the prior VERIFY command where nn was specified or by the editor entry default (established via your profile or the /SET LINESIZE).

Any of the above forms of the command will terminate the brief response mode of the editor (set by the BRIEF or VERIFY OFF commands) and set verify mode, causing the automatic display of lines changed or searched for by other edit commands.

If one of the following is specified:

```
VERIFY LONG
VERIFY LONG n1
VERIFY LONG n1 n2 n3
```

long verify mode will be set (for 3270s only) if brief or normal verify mode was in effect. If long verify mode was already in effect, it is retained. The long verify screen will be displayed. If any other operands (n1, n2, n3) are specified, the verify screen width, length and centering factor are altered as specified. If these parameters are not specified, the values as of the previous VERIFY LONG (where they were specified) are retained. If none had been specified previously, the defaults (72, 20, 7) are used.

If one of the following is specified:

```
VERIFY OFF
BRIEF
```

and verify long mode is in effect, the first VERIFY OFF or BRIEF command causes verify mode to be entered. The second VERIFY OFF or BRIEF command causes brief mode to be entered.

The current line (or long verify screen) can always be redisplayed by issuing the verify command with no operands. In brief mode lines that are changed in the file are not typed out. BRIEF mode affects commands such as BLANK, CHANGE, FIND, LOCATE, SEARCH, NEXT, OVERLAY and UP.

VERIFY FULL initiates the full screen editor and, optionally, sets the number of lines to be displayed prior to and including the current line. (It is not possible to invoke the full screen editor in a procedure.)

Examples:

1. Terminate the context editor and initiate the full screen editor for the member being edited.

   ```
   verify full 5
   ```

2. Set verify mode and cause 60 columns of the current line to be displayed after location and change commands.

   ```
   verify 60
   ```

3. Set long verify mode with 12 total lines, 5 of which are lines prior to the current line.

   ```
   verify long 72 12 5
   ```

4.

   ```
   line before request:          020 INPUT I,J,K
   request in verify mode:       ch /j,k/k,l
   response:                     020 INPUT I,K,L
   request:                      br
   request in brief mode:        ch /k,l/m,n
   response:                     (none)
   ```

Bibliography

Apart from the publications necessary for your work with assemblers or compilers, you may have to use the following IBM publications.

VSE/ICCF

VSE/ICCF Reference Summary, GX33-9011.
VSE/ICCF Introduction to Interactive Programming, SC33-6202.
VSE/ICCF Installation and Operations Reference, SC33-6203.
VSE/ICCF Messages, SC33-6205.
VSE/ICCF Licensed Program Specifications, GC33-6201.
VSE/ICCF Program Summary, GC33-6200.
VSE/ICCF Diagnosis Reference, LY33-9120.

VSE System

VSE/System Package, Messages and Codes, SC33-6181.
VSE/System Package, Diagnosis, SC33-6182.
VSE/Advanced Functions, System Management Guide, SC33-6191.
VSE/Advanced Functions, Diagnosis: Service Aids, SC33-6195.
VSE/Advanced Functions, Application Programming: Macro Reference, SC33-6197.
VSE/Advanced Functions, System Control Statements, SC33-6198.
Device Support Facilities, User's Guide and Reference, GC35-0033.
DOS/VS Sort/Merge V2 Programmer's Guide, SC33-4044.

VSE/POWER

VSE/POWER Remote Job Entry User's Guide, SH12-5328.
VSE/POWER Installation and Operations Guide, SH12-5329.
VSE/POWER Messages, SH12-5520.

Terminal Support

IBM 3270 Information Display System, Data Stream, Programmer's Reference, GA23-0059.

Index

VSE/Interactive Computing and Control Facility
Terminal User's Guide
Order No. SC33-6204-2

READER'S
COMMENT
FORM

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

____

____

____

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ________________

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page).

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

C33-6204-2

eader's Comment Form

Cut or Fold Along Line

Fold And Tape Please Do Not Staple Fold And Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1
180 Kost Road
Mechanicsburg, PA 17055

Fold And Tape Please Do Not Staple Fold And Tape

IBM®

VSE/Interactive Computing and Control Facility Terminal User's Guide
(File No. S370/4300-39) Printed in U.S.A. SC33-6204-2

VSE/Interactive Computing and Control Facility
Terminal User's Guide
Order No. SC33-6204-2

READER'S
COMMENT
FORM

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*
Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

____

____

____

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ________________

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page).

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

SC33-6204-2

Reader's Comment Form

Fold And Tape — Please Do Not Staple — Fold And Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1
180 Kost Road
Mechanicsburg, PA 17055

Fold And Tape — Please Do Not Staple — Fold And Tape

IBM®

Cut or Fold Along Line

VSE/Interactive Computing and Control Facility Terminal User's Guide
(File No. S370/4300-39) Printed in U.S.A. SC33-6204-2