

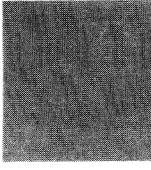
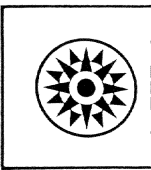
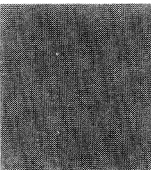
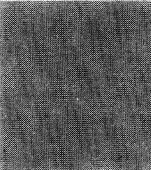
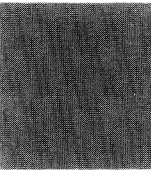
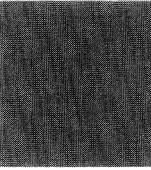
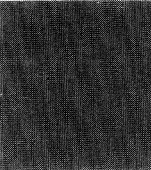
Systems Reference Library

Version 8.1

IBM System/360 Time Sharing System

FORTRAN Programmer's Guide

This publication describes how to use the IBM System/360 Time Sharing System (TSS/360) for compiling and executing programs written in the FORTRAN IV language. It also describes how to use the services and features of TSS/360 that, while not directly related to FORTRAN programming, are frequently of use to the FORTRAN programmer.



FIFTH EDITION (September 1971)

This is a major revision of, and makes obsolete, C28-2025-3 and Technical Newsletters GN28-3067 and GN28-3141.

This edition is current with Version 8, Modification 1, of the IBM System/360 Time Sharing System (TSS/360), and remains in effect for all subsequent versions or modifications of TSS/360 unless otherwise noted. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication, refer to the latest edition of *IBM System/360 Time Sharing System: Addendum*, GC28-2043, which may contain information pertinent to the topics covered in this edition. The *Addendum* also lists the editions of all TSS/360 publications that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing System/360 Programming Publications, Department 643, Neighborhood Road, Kingston, N. Y. 12401.

This publication is a guide to the facilities of the IBM System/360 Time Sharing System for the user of IBM FORTRAN IV. It is divided into three parts.

Part I is an introduction to the IBM System/360 Time Sharing System, directed to the user of the FORTRAN IV language. It contains basic information needed for effective use of the system and all of the information needed for effective use of Part II of this manual. Readers already familiar with the time-sharing system might profitably scan Part I and go directly to Part II.

Part II is a set of annotated examples. They begin with fundamental operations, such as logging on, and in succeeding examples progress to increasingly sophisticated concepts. The examples reproduce and comment on the user-system dialog as it would appear at a terminal with the exception that specific system response messages are not identified. The examples may be read for instruction; they may also be used as models for accomplishing common tasks.

Part III is a set of appendixes containing reference material for users who may need detailed information about the system.

Much of the material in the introduction and the appendixes duplicates or summarizes information in the examples (Part II) and other TSS/360 publications. Some material is unique, such as Appendix C, which gives guidelines for efficient programming and a discussion of the effects of compiler optimization on the use of the program control system (PCS).

Prerequisite Knowledge

Readers should be familiar with the IBM FORTRAN IV language, since this book does not describe the language, but rather describes the use of the language in the TSS/360 system.

The FORTRAN user will find the language specified in these publications:

IBM System/360 Time Sharing System: IBM FORTRAN IV, GC28-2007

IBM System/360 Time Sharing System: FORTRAN IV Library Subprograms, GC28-2026

If additional knowledge of the time-sharing system is needed, the following publications should be referred to:

IBM System/360 Time Sharing System: Concepts and Facilities, GC28-2003, provides a broader system survey than does this manual's introduction.

IBM System/360 Time Sharing System: Command System User's Guide, GC28-2001, describes the entire command language, including the program control system.

IBM System/360 Time Sharing System: System Messages, GC28-2037, lists all of the messages produced by the system.

IBM System/360 Time Sharing System: Terminal User's Guide, GC28-2017, gives details of the facilities and operations of the various terminals supported by TSS/360.

IBM System/360 Time Sharing System: Linkage Editor, GC28-2005, describes the linkage editor.

FORTRAN programmers who wish to use assembler language subroutines will need to be familiar with:

IBM System/360 Time Sharing System: Assembler Language, GC28-2000

IBM System/360 Time Sharing System: Assembler User Macro Instructions, GC28-2004

Contents

Part I. Introduction	1	Example 22: Survey of System Facilities and Some Housekeeping Methods	62
The System	1	Example 23: Generation Data Groups	65
Identifying You to the System	1	Example 24: Tape and Disk Medium Transfers of Virtual Access Method Data Sets	67
Conversational Use of the System	1	Example 25: The Text Editor Facility	68
Conversational Task Initiation	2	Example 26: The Text Editor Facility	69
SYSIN and SYSOUT	2	Example 27: Use of Procedure Definition (PROCDEF) ..	71
Virtual Storage	2	Example 28: The User Profile Facility	72
Sharing Time	3		
System Catalog	3		
Terminal Session Activity	4		
Entering Commands	4		
Compiling and Running a Program	4		
Checking Out and Modifying Programs	5		
Planning Problem Program Input/Output	5		
Specifying TSS/360 Problem Program I/O	6		
Data Sets with Virtual Storage Organization	6		
Physical Sequential Data Sets	7		
Generation Data Groups	7		
Data Set Definition	7		
Cataloging and Uncataloging Data Sets	8		
Using System Storage	9		
Protecting and Sharing Data Sets	9		
Maintaining Program Libraries	10		
Copying, Modifying, and Erasing Data Sets	12		
Conversational Task Termination	12		
Nonconversational Use of the System	12		
Nonconversational Task Initiation	12		
Nonconversational Command Procedure Processing	13		
Nonconversational Task Termination	13		
Mixed Mode Use of the System	13		
Remote Job Entry	13		
Command Directory	14		
Part II. Examples	19		
Example 1: Initiating and Terminating a Conversational Task	20		
Example 2: Compilation and Correcting from the Terminal	22		
Example 3: Compilation and Correction from the Terminal	24		
Example 4: Compile and Run	26		
Example 5: Correcting and Recompiling a Prestored Source Program	28		
Example 6: Writing a Data Set and Printing It	30		
Example 7: Reading and Writing Cataloged Data Sets ..	32		
Example 8: Multiple Compilation Before Execution	34		
Example 9: Use of PCS Immediate Statements	36		
Example 10: Use of PCS Dynamic Statements	38		
Example 11: Input and Output on Tape	40		
Example 12: Conversational Initiation of Nonconversational Tasks	42		
Example 13: Preparing a Job for Nonconversational Processing	44		
Example 14: Storing DDEF Commands for Later Use	45		
Example 15: References to Subroutines	47		
Example 16: Entering Data for Later Use	49		
Example 17: Data Set Considerations when Interrupting a FORTRAN Execution	52		
Example 18: Sharing Data Sets	54		
Example 19: Manipulation of Several Forms of a Program	56		
Example 20: Terminal Input of a Pre-Punched Program for Compilation and Running	58		
Example 21: Intra-task Carryovers	60		
		Part III. Appendixes	
		Appendix A. Use of the FORTRAN Compiler	74
		Entry and Correction of FORTRAN Source Statements ..	74
		Format of Source Lines	74
		Card Format Line (Both Nonconversational and Console Card Reader)	74
		Character Sets — Card Format	74
		Keyboard Format	74
		Character Sets — Keyboard Format	75
		Mixed Card and Keyboard Input	75
		Efficient Correction Techniques	76
		Entry of Keyboard Source Statements for Later Punching and Recompilation	78
		Compiler Diagnostic Action	78
		Compiler Options and Listings Produced	80
		FORTRAN Parameters	80
		Explicitly Defaulted	80
		Implicitly Defaulted	80
		Structure and Description of Compiler Listings	82
		Heading Page	83
		Source Program Listing	83
		General Description of Output Module Listing	83
		Default Option Listing	83
		Detailed Description of Output Module Listing	85
		Description of PSECT Listing	87
		Description of Table of Initialized Variables	89
		Description of Symbol Table Listing	90
		Description of Cross Reference Listing	91
		Description of Storage Map Listing	91
		Destination of Compiler-Produced Listings	92
		Conversational Tasks	92
		Nonconversational Tasks	92
		FORTRAN IV Library Subprograms: Indirect References	92
		Reference to Subroutines	92
		Destination of Output	93
		Compiler Restrictions	93
		Appendix B. PCS and FORTRAN Object Programs	97
		General	97
		Commands and Statements	97
		Sequence of Operation	97
		Conversational Mode	98
		Nonconversational Mode	98
		Notation	98
		Directives	98
		Operators	98
		Symbols	99
		FORTRAN Statement Numbers	99
		Subscripted Symbols	100
		Constants	100

Expressions	100
Arithmetic Expressions	100
Logical Expressions	101
Ranges	101
Commands	101
QUALIFY Command	101
AT Command	101
DISPLAY Command	102
DUMP Command	102
IF Command	103
REMOVE Command	103
CALL, GO, BRANCH Commands	104
SET Command	104
STOP Command	105
PCS Diagnostics	105
Dimension Errors	105
Range Errors	105
Program Interruption	105
Dummy Arguments	106

Appendix C. Programming Considerations	107
Object Time Efficiency	107
Object Code Optimization	107
Compiler Optimization	107
Efficient Use of FORTRAN Statements	108
Use of Linkage Editor to Improve Object Time Efficiency	109
Use of Dynamic Loader to Improve Object Time Efficiency	110
Use of Control Section Packing to Improve Object Time Efficiency	110
Effect of Compiler Optimization on PCS Usage	110
Multiple Executions	111
Data Definition Considerations	111
Linking COMMON between Multiple Executions	111
Program Libraries	112
Program Library List Control	112
Substituting FORTRAN IV-Supplied Subprograms	113
Sharing Libraries	113
Recovering from Errors when Dynamically Loading	114
Shared Code (PUBLIC) Considerations	115
System Naming Rules	116
User-Assigned Names	116
Reserved Names	116
External Symbols	116
Reserved Names Associated with Data Sets	116
Compiler-Assigned Names	116
Miscellaneous Programming Considerations	117
Floating-Point Computations	117
Object Program Interrupt Provisions	117
STOP/PAUSE/RETURN Differences	117
Link-Editing FORTRAN Programs	118
Use of RUN Command and Call Statement with FORTRAN Subprogram Module Names	118
Initial Content of FORTRAN Variables	118

Appendix D. Assembler Language Subprograms	119
FORTRAN Object Program Structure	119
Subprogram References	119
Proper Register Usage	120
Reserving a Parameter Area	120
Reserving a Save Area	120
Variable-Length Parameter Lists	120
Types of FORTRAN Calls	121
Linkage between FORTRAN and Assembler Language Programs	121
CALL where the Argument is a Variable Name	121
CALL where the Argument is a Subprogram Name	122
Using Data in COMMON	123
Referring to Variables in an Array	123

Appendix E. Specification of Data Set Characteristics	124
Data Set Creation and Structure	124
Access Methods	124
Virtual Access Method	124
Physical Sequential (PS)	125
Data Set Records	126
Variable-Length Format	126
Fixed-Length Format	126
FORTRAN Records	127
Formatted Records	127
NAMELIST Records	129
Unformatted Records	130
Summary of FORTRAN Data Set Formats	130
FORTRAN Operations on Data Sets	130
Generation of New Data Sets	130
Reading Existing Data Sets	132
From Outside TSS/360	132
From Other than FORTRAN Programs on TSS/360	132
From FORTRAN Programs on TSS/360	132
Exception Handling	132
Positioning Statements and Sequence Rules	133
Execution I/O Error Messages	133
SECURE Requirements for Nonconversational Tasks	133
Guide to DDEF Commands	134
Basic DDEF Command	134
DDNAME = FTxxFyyy	134
DSORG =	135
DSNAME =	135
Default of DDEF Commands	135
Conversational	135
Nonconversational	136
Full DDEF Command	136
DDNAME	137
DSORG	137
DSNAME	138
UNIT	138
SPACE	138
VOLUME	138
LABEL	139
DISP	139
OPTION	139
RET	140
DCB	140
DDEF Summary	142
Sample DDEF Commands	142
Error Messages for the DDEF Command	145
Data Set Definition Rules for Language Processing	145
Data Set Definition Rules for TSS/360 Commands	145

Appendix F. Attention Considerations	149
Interrupting Execution	149
Interrupting Privileged Commands	149
Interrupting Nonprivileged Commands and User Programs	149
Attention Levels	149
Using the Program Control System (PCS) with ATTENTION	149
Responding to Attention Interruptions	149

Appendix G. Command Formats	151
Operands	151
Metasymbols	151
General Forms	152

Appendix H. Carriage and Punch Controls	162
--	-----

Appendix I. Sample Program	163
Part One - Nonconversational	163
Part Two - Conversational	165

Figures

FIGURE	TITLE	PAGE	FIGURE	TITLE	PAGE
1	System Catalog	4	14	Table of Initialized Variables	89
2	A Simple Compile and Run	5	15	Symbol Table Listing	89
3	Data Set Identification, FORTRAN Programs	7	16	Cross-Reference Listing	90
4	Catalog Example	9	17	Storage Map Listing	91
5	Sharing of Cataloged Data Sets	11	18	Save Area Format and Word Content	120
6	Nonconversational Task Initiation	12	19	Maximum Record Lengths (Bytes)	126
7	FORTRAN Parameters	80	20	Record Formats – Virtual Sequential	127
8	Compiler Parameters Default and Prompting Description	81	21	Record Formats – Virtual Index Sequential	128
9	Heading Page	83	22	Record Formats – Physical Sequential	129
10	Source Program Listing	83	23	Basic DDEF Command	134
11	CSECT and PSECT Listings for Default Listing Options	84	24	Full DDEF Command	137
12	CSECT Listing	84	25	Card Listing for CONV	163
13	PSECT Listing	86	26	Compilation Listing for CONV	164
			27	SYSOUT Listing for CONV	165
			28	Conversational SYSIN-SYSOUT for CONV	165

Tables

TABLE	TITLE	PAGE	TABLE	TITLE	PAGE
1	Command Directory	14	10	Data Set Format Summary	131
2	Compiler Diagnostic Action	79	11	DDEF Parameter Requirements by Data Set Type	143
3	Destination of Compiler Output	95	12	Data Set Definition Rules for Language Processing	146
4	Simple Source Program Restrictions	95	13	Data Set Definition Requirements for Commands	146
5	Complex Source Program Restrictions	94	14	Responding to Attention Interruptions	150
6	Shared Data Set Commands	114	15	Carriage Control Characters	162
7	STOP/PAUSE/RETURN Differences	118	16	Punch Control Characters	162
8	Linkage Registers	120			
9	Dimension and Subscript Format	123			

In the Time Sharing System/360 you can run a program *conversationally*: you and the system can exchange information during the entering and execution of your program.

To compile a program conversationally, you enter it at a typewriter-like terminal. The system analyzes each program statement as it is entered. If the system finds an error, it tells you so and offers you a chance to correct it. When the whole program has been entered, it is analyzed as a whole, and again you can correct any errors the system may find. Then you can execute the program and monitor its progress. For example, you can intervene during execution to check on the current value of a variable, or leave a flag to the system to stop execution should a specified condition arise. You can even make temporary patches to correct program troubles.

You can also run a program *nonconversationally* — for instance, when the program has been checked out and you know it will run satisfactorily, or when you cannot stay at the terminal to converse with the system. Nonconversational (background) processing in tss/360 works much like batch processing in other IBM systems.

You can run in mixed mode — that is, you can start a program conversationally and switch to nonconversational processing. Once a program is running nonconversationally, however, you may not switch back to conversational processing.

The System

tss/360 is a special set of programs that has been designed to make it easier for you to use a computer:

- A supervisor program controls the overall operation of the system, and provides the time sharing environment that lets a number of users employ the system concurrently.
- A group of service routines perform program control and data management functions for each user, as well as for the system.
- A third set of programs allows you to compile and develop your problem programs.

This publication explains how to use these programs, without involving you in their structure or their detailed internal operations.

Identifying You to the System

Before you first use tss/360 you must be granted access to the system by either your system administrator or your system manager.¹ They, in effect, join you to the system by storing the following information about you:

- *User Identification (userid)* — a code that uniquely identifies you to the system.
- *Password* — a code word used in validating your attempt to get on the system under the above userid. The password is a further protection against unauthorized use of the system or unauthorized use of your data sets or charge number.
- *Charge Number(s)* — account number(s) against which your use of the system is charged.
- *Priority* — a code indicating the relative priority of your work in the system.
- *Privilege Class* — a code identifying you as a user, i.e., an individual who can employ the special set of commands reserved for users (as opposed to the commands reserved for, say, the operator).

From the information supplied by your manager or administrator, the system can recognize you, and validate your use of the system when you wish to begin processing. This information remains in the system until your system manager or administrator withdraws your right to use the system.

Conversational Use of the System

In conversational processing, you communicate with the system by means of a terminal. The terminal is a typewriter-like device. One type, the IBM 2741, is an IBM Selectric typewriter specially equipped for terminal use; another type, the IBM 1050 System, can include both a typewriter and a card reader. With the 1050 you can enter input into the system via the keyboard or the card reader. Your terminal may be located at the computer installation or at a remote location. In any event, all terminal operation is much the same: you enter a command directing the system to do certain work, the system responds, you enter another command, etc. You don't have to be an expert typist; correcting typing errors is a straightforward process, as shown in the examples.

¹If you are interested in additional details on system management and administration, refer to *Manager's and Administrator's Guide*.

You will find that you do not require extensive computer training to use TSS/360. You must know three things:

- *The procedure for setting up your terminal for operation.* This is a matter of setting a few switches. This manual does not discuss the procedures and settings for the various terminals — see *Terminal User's Guide* or ask someone to show you the correct procedure for setting up your terminal.
- *The TSS/360 FORTRAN IV Language*, the language in which you express your problem-solving procedure. This language is used for illustration throughout this publication; it is explained in detail in *IBM FORTRAN IV*. In TSS/360, you also have a variety of mathematical and service subprograms available for your use. These are described in *FORTRAN IV Library Subprograms*.
- *The TSS/360 Command System*, involving the commands you will use to converse with the system. Almost every command is shown in the examples in Part II of this manual. Many typical uses are shown, but not every use of every command. Should you need more information than is in the examples or the appendixes, consult *Command System User's Guide*, which describes the commands in detail. The commands are explained briefly at the end of this introduction.

In conversational mode, you engage in dialog with the system. The system responds to your requests, confirms actions, and informs you of any errors. Complete details on system response messages are presented in the *System Messages* publication.

The work done between logging on and logging off is called a *task*. You may run one or many programs as part of a single task. The work you do on a task at a terminal is called a *terminal session*. Since a task may begin conversationally but end nonconversationally, task is not necessarily synonymous with terminal session.

Conversational Task Initiation

You use the following procedure to initiate conversational processing:

1. Make certain the terminal is set up for operation under TSS/360 (proper switch settings, power on, etc.)
2. Either:
 - a. Dial up the system, if it has a telephone-like modulator/demodulator (modem). The phone number is determined by your installation.
 - b. Press the attention button on the terminal, if the terminal is "hardwired" (i.e., directly connected to the computer).

When you press the attention button or dial up the system, you begin the log-on process and set up a conversational task in the system. If you have been granted access to the system, and identify yourself properly in the LOGON command in accordance with the parameters set up for you at join time, the system completes the initiation of your task. (If you cannot log on, you should notify your system manager or system administrator.)

SYSIN and SYSOUT

From your point of view, initiating a task means that the system has prepared itself to perform work for you. You can now converse with the system as if you alone were using it. You have unique communication paths in the system, permitting it to read from and write to your terminal independently of all other tasks. You can thus define work for the system by issuing commands, and the required programs and data will be loaded into main storage and processed, as you specify, regardless of the work other users may be simultaneously specifying.

Your task's input to the system contains the sequence of commands you issue; this sequence is called SYSIN. Your system input stream can also include data to be prestored in the system, or actual input records to an executing program. When you are in the conversational mode, your terminal is your task's SYSIN device. Your task's system output stream, called SYSOUT, is directed to the terminal. It consists basically of system messages; it may also contain output from your programs if you so choose. Because the terminal is thus a combined SYSIN/SYSOUT device, the terminal listing will contain a mixture of the two system streams.

You and every other user have your own unique SYSIN/SYSOUT. You also have the following:

- Your own virtual storage space
- A scheduled time interval in which your task is executed
- Your own catalog

Virtual Storage

In TSS/360, you are not directly concerned with the physical limitations on main storage. Special addressing techniques, internal to the system, provide you with a storage capacity theoretically equal to the total range of addresses that can be specified in an instruction. The system's addressing techniques effectively combine sections of main and secondary storage, creating a virtual storage area in which your task operates. Your installation will inform you of specific virtual storage limits on your problem programs and data.

Although you have large virtual storage capacity, efficient programming is important; performance can

be degraded by excessive demands on the available storage at an installation.

When you log on, the system routines essential to your task are loaded into your virtual storage. These routines are a permanent part of your virtual storage, i.e., they remain there throughout your task.

You obtain other system routines by issuing commands and executing programs. These routines are loaded into, and unloaded from, your virtual storage on a demand basis.

You control the residence in your virtual storage of the linkage editor and your problem programs. (Refer to the LOAD, LNK, RUN, CALL, GO, and UNLOAD commands in the table at the end of this introduction.)

An important aspect of TSS/360 virtual storage management is the protection it provides. Each user has his own storage space for program execution. Another user cannot interfere with your executing programs, nor can you interfere with his, because neither of you can refer to the other's virtual storage space.

Sharing Time

Others may be using the system at the same time you are. The system appears to be serving each of you exclusively because it is repetitively giving each of you a time slice, or an interval, during which all the facilities required by your task, including computer execution time, are in fact exclusively yours. Unless the system is overloaded, its speed will allow it to do your work as well as that of other users without the intervals being apparent to you.

TSS/360 can also operate with several terminals sharing a single task. This mode of operation is not discussed in this publication; refer to *IBM System/360 Time Sharing System: Multiterminal Task Programming and Operation*, GC28-2034 for a description.

System Catalog

Conceptually, the system catalog is very much like the catalogs used in libraries. It is an index that points to items that reside elsewhere. You use it initially to record the location of data, so that you don't have to keep track of where the data is located and so that you can later retrieve the data by its name alone. The structure of the catalog protects your data sets from being accessed by other users, unless you specifically permit others to share them.

To understand the structure and significance of the system catalog, you must become familiar with the basic concepts of data set, data set name, and data set residence.

A *data set* is a named collection of one or more records. For example, all of the following are data sets: a source program, a library of compiled programs, the

collection of FORTRAN input records needed by a program.

A *data set name* uniquely identifies a data set. It is in the form of one or more symbols separated by periods. For example, ROCKET.TESTFIRE.APRIL14. Each symbol can consist of from one to eight alphameric characters, the first of which must be alphabetic. Starting from the left, each symbol of the name is a category within which the next symbol is a unique subcategory. A fully qualified name identifies an individual data set. A partially qualified name identifies a group of data sets. For example, if ROCKETS.TESTFIRE.APRIL14 is a fully qualified data set name, ROCKETS and ROCKETS.TESTFIRE are partially qualified names identifying groups of data sets, one of which is ROCKETS.TESTFIRE.APRIL14. The group ROCKETS.TESTFIRE is a subgroup of ROCKETS.

For example, examine the gross structure of the system catalog illustrated in Figure 1, and note the following:

1. The system catalog consists of a master index and sets of subordinate catalog entries. It is, in effect, a collection of *separate* catalogs. The system has its own catalog and each user has his own catalog.
2. The various catalogs are an index of the data sets associated with them. Data sets that are to be cataloged must reside on one or more direct-access or magnetic tape volumes. A volume can be a removable disk pack or a tape reel. Some direct-access volumes are public, meaning that they are permanently mounted while the system is running, and they can be accessed by all users. Some direct-access volumes, and all magnetic tape volumes are *private*. This means that they are not mounted on the system until needed, that they are demounted when no longer needed, and that they can be used by only one user at a time. Data sets on either public or private volumes can be cataloged.

When the system was generated at your installation, all catalog entries for system data sets were created, including SYSLIB, which contains the system routines that are loaded on demand — for example, the FORTRAN-supplied subprograms.

Your master index entry in the system catalog is created when your system manager or administrator joins you to the system. At that time, your user identification is placed in the master index and another special entry is created in your catalog for a data set called your USERLIB. Your USERLIB is your own private library for object programs.

Except for USERLIB, you control all entries in your catalog by the way you name your data sets and by the way you use the cataloging and uncataloging fa-

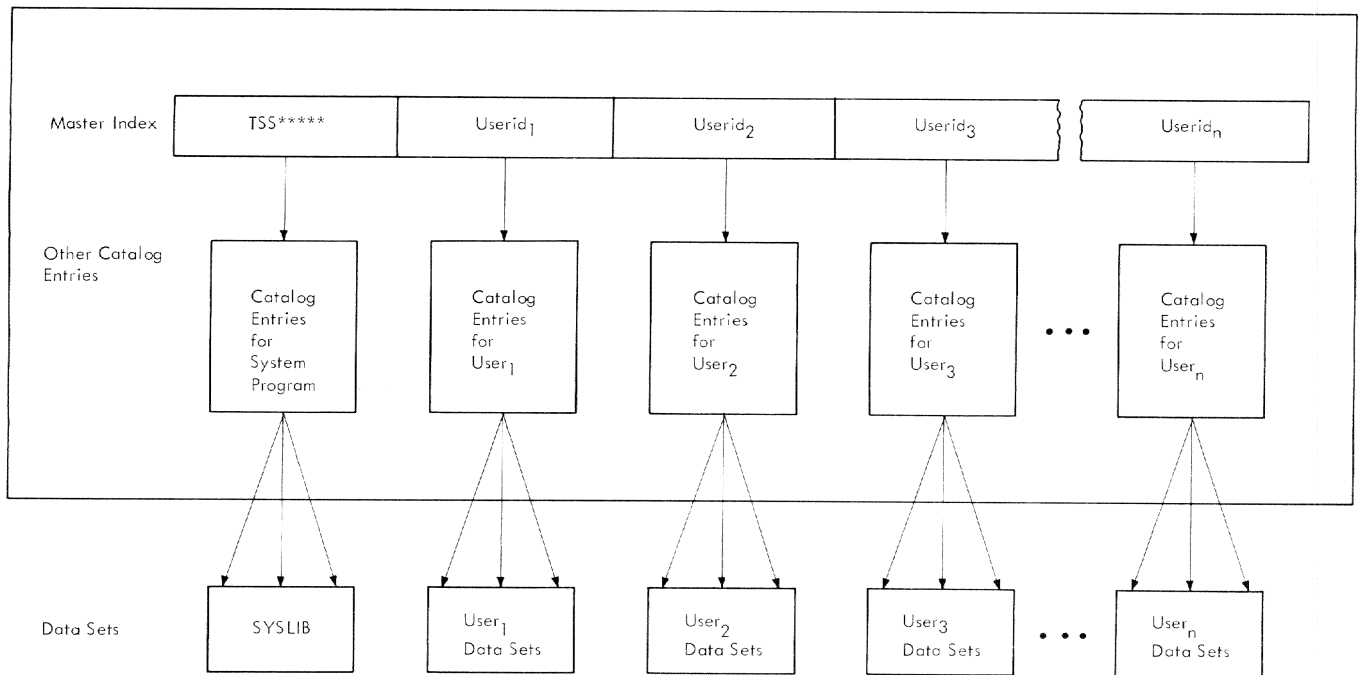


Figure 1. System Catalog

ilities of the system. Some of these facilities are for entering, removing, and renaming catalog entries. Others are for indicating which data sets can be shared by others and to what extent. These facilities are described later in this section. The key points here are these:

- Your catalog exists in the system from the time you are joined until the time your access privilege is withdrawn.
- Cataloging data sets is the only means of retaining data sets on public volumes from session to session. It also simplifies later use of those data sets.
- You can share your programs and data with others or not as you wish.

Terminal Session Activity

Entering Commands

To enter commands, you simply type in the required characters and press the return key on the terminal. What you type in, of course, depends on what you wish to do and the content of the commands required. Each command has an operation part specifying what is to be done (as `RUN`), and each may have one or more operands that qualifies the operation (as `LOC=` followed by the name of your object program, say `MAIN`. This qualifies the operation to mean "execute my object program, `MAIN`").

If you enter an incorrect command, the system issues a message which informs you of the error. The system also issues messages that give you information helpful in assessing the system's activity relative to your task. System messages are issued automatically as the conditions causing them arise.

Compiling and Running a Program

Suppose you wanted to simply compute sine `A`, for a single value of `A`, and print the result at the terminal. You might design the following source program to do this:

```
A=.2
SINE=SIN(A)
10 FORMAT ('THE SINE OF' ,F7.4, 'IS' ,F7.4)
WRITE(6,10)A,SINE
STOP
END
```

You could then compile and run that program by the conversational task illustrated in Figure 2. The `LOGON` and `LOGOFF` commands are used to initiate and terminate the task. `FTN` initiates FORTRAN compiler processing. To control the compilation you specify a number of parameters: name of the compiled program, listings you want, etc. The compiled program is automatically stored in your `USERLIB` as an object program module. You execute it by the `CALL` or `RUN` command to obtain your result.

Log on	1. Press the attention button or dial up system. 2. Issue a LOGON command.	System makes you an active user.
Compile	1. Issue a FTN command (FTN is the name of the FORTRAN compiler). 2. Enter the parameters required to control the compilation (including the name by which you want to identify the object program). 3. Enter your source program.	System compiles your source program; it then stores your object program for you (in your USERLIB).
Run	1. Issue a CALL PGM command (where PGM is the name you assigned your object program at compile time above).	System retrieves your object program, executes it, and prints the result on your terminal: THE SINE OF 0.2000 IS 0.1987.
Log off	1. Issue a LOGOFF command.	System terminates your task and releases your terminal.

Figure 2. A Simple Compile and Run

In TSS/360 your source programs can use many of the system-supplied subprograms. For example, in the program illustrated in Figure 2, you used the SIN subprogram. These programs reside in SYSLIB and are available during execution when your program invokes them. Similarly, you can design and compile your own FUNCTION and SUBROUTINE subprograms, store them in your USERLIB (or some other library) and use them during later program executions.

Checking Out and Modifying Programs

The FORTRAN compiler includes conversational prompting and diagnostic facilities that assist you in debugging your source program. It also includes optional facilities for storing and cataloging your source and listing data sets, and for including an Internal Symbol Dictionary (ISD) in your object module. An ISD allows you to make full use of the Program Control System (PCS).

You can use PCS commands and statements to perform one, or any combination, of these:

1. Request display of data fields and instruction locations within your program, specifying these items by their symbolic names as used in the source language program.
2. Modify variables within your program, specifying these variables by their symbolic names and specifying the new value for each variable.

3. Specify the statements within your program at which execution is to be stopped or started. When program execution has been stopped, you may intervene, as described in items 1 and 2, before you direct resumption of program execution.
4. Specify the statements within your program at which the actions described in items 1 and 2 are to be automatically performed.
5. Obtain the values of your program's variables at a specified point in its execution, with the variables formatted according to their types.
6. Establish logical (true or false) conditions that allow or inhibit the actions described in items 3, 4, and 5.

The use of Program Control System facilities does not impose any restrictions on your source coding. In general, the use of program control facilities will greatly simplify the preparation of source programs, because many functions previously source-coded can conveniently be made available after compilation. PCS is discussed in greater detail in Appendix B.

Planning Problem Program Input/Output

In most TSS/360 installations, a problem program does not communicate directly with unit record devices (card reader/punches and printers). You organize input/output data flow as follows:

1. *Prior to program execution*, you store input data in the system on a direct-access device. If the data is the output of a previously executed program, you can simply write it on public storage during that program so that it will be retained for subsequent use. If the data involved is new input, you can pre-store it using the following facilities:
 - Text editor facilities
 - DATA command
 - Operator procedures, involving your card input deck or magnetic tape reel
2. *During program execution*, you will generally READ input data from the direct-access device on which you stored it; and you will WRITE output data to a direct-access device (for later actual output, following execution). However, in TSS/360, you also have the following additional facilities for input/output during program execution:

Input: You can READ a record dynamically from your terminal, READ input data from the system's tape devices; or SET data in your program (based on program conditions) if you like.

Output: You can WRITE a record to your terminal, WRITE output data on magnetic tape; and PAUSE in your program, print out data or a message at your terminal, and return control to the terminal for insertion of additional commands and then continue processing.

You can also STOP your program and DISPLAY the final results of its computation.

3. *Following execution*, you can print out or punch on cards the program output you stored on a direct-access device, using the PRINT and PUNCH commands, respectively. You can also produce a magnetic tape for subsequent printing by issuing a WT (write tape) command.

Since you can communicate with your programs during their execution, you can design programs that do not require use of conventional I/O devices; all I/O can be achieved via the terminal. For example, you can design your programs so that when predetermined events occur, intermediate results are printed out at your terminal. You can then decide how you want to proceed: supply additional or different data at that time; change the sequence of program execution; stop the programs; examine key final results prior to initiating their final printout; etc.

Specifying TSS/360 Problem Program I/O

In TSS/360, to specify problem program I/O activity, you must consider both of the following:

1. Use of appropriate FORTRAN I/O statements in the source program to indicate the data transfer or I/O control functions.
2. Use of (or omission of) DDEF (define data) commands to identify the name, location, organization, etc., of the data sets associated with the FORTRAN I/O statements.

Data Transfers: You use the FORTRAN READ and WRITE statements to transfer data to and from your program. You should regard a data set as a continuous string of data which you have subdivided for separate processing by your FORTRAN program; the subdivisions are termed FORTRAN logical records. FORTRAN logical records are defined by one of the following:

1. A FORMAT statement and a list referred to by an I/O statement (formatted records).
2. An I/O list appearing in an I/O statement that does not contain a reference to a FORMAT statement (unformatted records).

3. A NAMELIST name appearing in an I/O statement (NAMELIST records).

You define the overall relationship of a data set's records by specifying the data set's organization. (You do this in the DDEF commands discussed later in this section.) In TSS/360, there are two fundamentally different types of data set organizations: virtual storage data sets and physical sequential data sets.

Data Sets with Virtual Storage Organization

Data sets with a virtual storage organization can reside only on direct-access volumes. You process these data sets on the basis of the records they contain. Virtual storage data sets can have any of these specific organizations:

1. *Virtual sequential:* This is the standard FORTRAN I/O data set organization; the term vs is used to describe it. In a vs data set, the order of the logical records is determined solely by the order in which the records were created. In creating this type of data set, you provide the system with a stream of records. The system organizes the data into pages, and stores the data set on a direct-access device. After the data set has been created, you can read back the records in the order in which they were created merely by requesting one record after the other.
2. *Virtual index sequential:* A data set with this organization is referred to as a vi data set. As a FORTRAN user, you will probably use vi data sets only when interfacing with programs written in assembler language that require this organization. In a vi data set, the records are organized in sequence based on a data key associated with each record. During FORTRAN program execution, you can create and read vi data sets sequentially, but you cannot use the random-access capabilities of this organization.

There are two special types of vi data sets — line data sets and list data sets. A *line data set* is one that is organized by line number, where each line is a record and is prefixed with the line number as its key. Source programs are line data sets. You can inspect and display these data sets by line number using the LINE? command. Other commands enable you to effect replacements, insertions, and deletions on line data sets. *Note:* Records in a line data set must be variable-length (format V); fixed-length (format F) records are not permitted.

A *list data set* contains the listings produced as output by the system's language processors; it is organized by line number, where each print line is a record and is suffixed with a line number as its key.

In conversational mode, printing of language-processor listings is not automatic; you can have a listing printed only if you issue a PRINT command.

3. *Virtual Partitioned.* A virtual partitioned data set, referred to as a VP data set, is used to combine individually organized groups of data into a single data set. Each group of data is called a member, and each member is identified by a unique name. Program module libraries are a good example of a VP data set. Your USERLIB is organized this way, and the compiled program modules you store in USERLIB are its members.

The partitioned organization allows you to refer to either the entire data set (via the partitioned data set's name) or to any member of that data set (via a name consisting of the name of the data set qualified by the member name in parentheses).

The partitioned data set may be composed of VS or VI members or a mixture of both. Individual members, however, cannot be of mixed organization.

Physical Sequential Data Sets

Data sets with a physical sequential organization can reside on either direct-access or magnetic tape volumes. The logical records in these data sets have an organization which is determined solely on the basis of their position relative to the beginning of the data set. When these records are processed in rss/360, the block is used as the unit of transfer to and from the device involved. A block can consist of one or more logical records. Data sets with physical sequential organization are called PS data sets. You will use PS data sets each time you process magnetic tape in your programs. Volumes containing data sets with PS organization can be interchanged among rss/360 and IBM System/360 Operating System installations.

Generation Data Groups

The cataloging facilities of rss/360 provide an option that assigns numbers to individual data sets in a sequentially ordered collection, thereby allowing you to catalog the entire collection under a single name. You can distinguish among successive data sets in the collection without assigning a new name to each data set. Because each data set is normally created from the data set created on the previous run, the new data set is called a generation, and the number associated with it is called a generation number. The entire structure of data sets of the same name is called a generation data group (GDG). You can refer to a particular generation by specifying, with the common name of the group, either the relative generation number or the absolute generation name of the data set.

Data Set Definition

In a FORTRAN I/O statement, the data set referred to is identified by an unsigned integer constant or integer variable whose value may be any number from 1 to 99. The relationship between a data set reference number and the actual data set is provided in the DDEF command. This I/O technique provides you with a degree of device independence; you do not need to change your program if the residence of data sets it processes changes from one execution to another.

The basic method used to identify FORTRAN data sets is illustrated in Figure 3. The system first relates the data set reference number of the READ or WRITE statement to the ddname operand of the corresponding DDEF command. For FORTRAN data sets, the ddname operand of the DDEF command is always of the form FTXXFyyy, where FT indicates FORTRAN, XX is the data set reference number, and yyy is a FORTRAN sequence number used to differentiate data sets, i.e., in a sequence of data sets which are to be referred to by the same data set reference number during the course of program execution (at any one time, the data set reference number refers to one data set only). Having

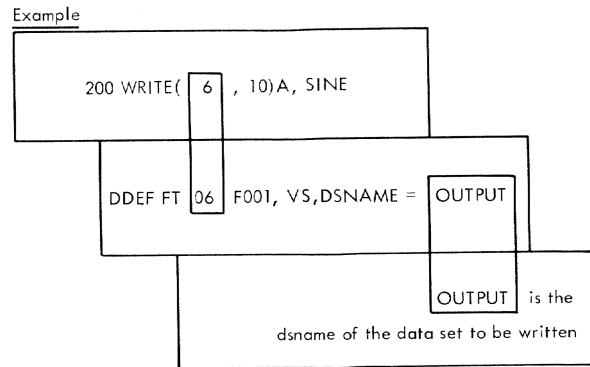
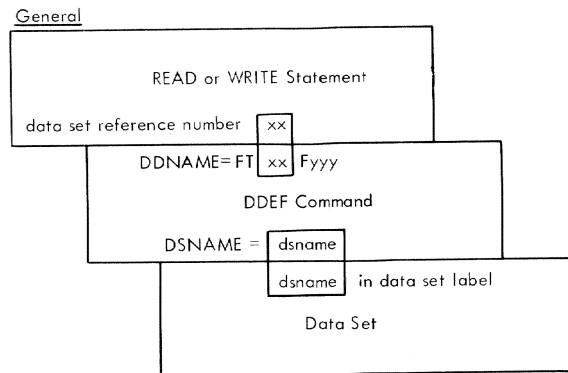


Figure 3. Data Set Identification, FORTRAN Programs

found the corresponding DDEF command, the system then obtains the name of the data set from the *dsname* operand of that DDEF command. Other information in the DDEF command (or already in the system, if the data set is cataloged) is then used to determine such things as: data set residence, i.e., where the data set is (input data set) or is to be placed (output data set); organization of the data set; routines necessary to process the data set; etc.

All object time FORTRAN data sets except those in SYSIN and SYSOUT require a DDEF command. In both conversational and nonconversational modes, if you omit a DDEF command normally associated with a READ or WRITE statement, the system will default to SYSIN or SYSOUT. There are also FORTRAN statements which automatically assume SYSIN or SYSOUT (STOP, PAUSE, and several I/O statements from previously implemented FORTRAN systems that were retained in TSS/360 FORTRAN). In a conversational task, SYSIN and SYSOUT involve your terminal. In a nonconversational task, you define the SYSIN data set (or submit it to the operator) and the system defines SYSOUT.

The DDEF command has other uses besides defining the data sets used during execution of a program. You can also use it to define the data sets used by certain commands, to define job libraries, to define a special data set (called PCSOUT) for the DUMP program checkout command, and to concatenate input data sets (i.e., relate them so that several different data sets can be read in as if they were a single data set).

Any DDEF command you issue during a task remains in force throughout the task, unless you enter a RELEASE command for that data set. The RELEASE command is the opposite of the DDEF command: the DDEF command sets up task control information for the data set; the RELEASE command removes that information.

The DDEF commands used in a session or in a command procedure need not be issued directly during the session or be included explicitly in the command procedure. One, or more, or all, of the DDEF commands needed can be made available by using the CDD (call data definition) command.

The CDD command is used to retrieve one or more DDEF commands from a data set; you must supply the name of the data set. If this is all you specify, the system assumes that you want to use all the DDEF commands in the data set. If you want to use only selected DDEF commands, you identify each by its ddname. You should prestore frequently used DDEF commands in a data set and call them in this fashion wherever possible. CDD can be used in either conversational or nonconversational tasks.

In a conversational task, the system analyzes the data set's requirements at the time the DDEF command is issued. It will then attempt to allocate the required resources (and, for private volumes, issue any mounting messages that are required) at that time. If the required space cannot be allocated, or the specified volumes cannot be mounted, the system will inform you, thereby allowing you to proceed with other work.

The DDEF command is illustrated in the examples, and is discussed in detail in Appendix E.

Cataloging and Uncataloging Data Sets

You can catalog and uncatalog data sets in several ways. Sometimes cataloging is automatic; in other cases, you must issue a CATALOG command to catalog the data set. All data sets with virtual storage organization that reside in public storage are automatically cataloged when they are created.

The CATALOG command sets up the catalog entry for the named data set. For example, suppose you are user JOHNDOE and you want to catalog a data set named ENG.PHYSICS.TEST2. If you issue a CATALOG command naming that data set, the system establishes entries in your catalog, as shown in Figure 4.

1. From your user identification, the system locates your catalog in the system catalog (via the master index).
2. It then sets up any indexes needed for each level of qualifier in your data set name. (Some of these may already exist.)
3. When it has established the lowest level index (in this case, TEST2), it records in the catalog the specific volume on which the beginning of the data set is located.

The CATALOG command can also be used to alter the entry of a previously cataloged data set; i.e., you can rename a cataloged data set. If you employ generation data groups (GDG), you must initially use the CATALOG command to set up the structure for the GDG: name, number of generations to be retained, disposition of old generations when the specified number of retentions is exceeded, etc. Then you can use the DDEF command to define new data sets as generation levels of the GDG, or you can use the CATALOG command with the NEWNAME operand to rename an existing data set as a generation level.

When you catalog a data set, you can specify either read-only or unlimited access. You can always erase your own data set, but if you have cataloged it with read-only access, you cannot write into it, thus ensuring against accidentally overlaying data.

You can use the DELETE command to remove a catalog entry for a data set if:

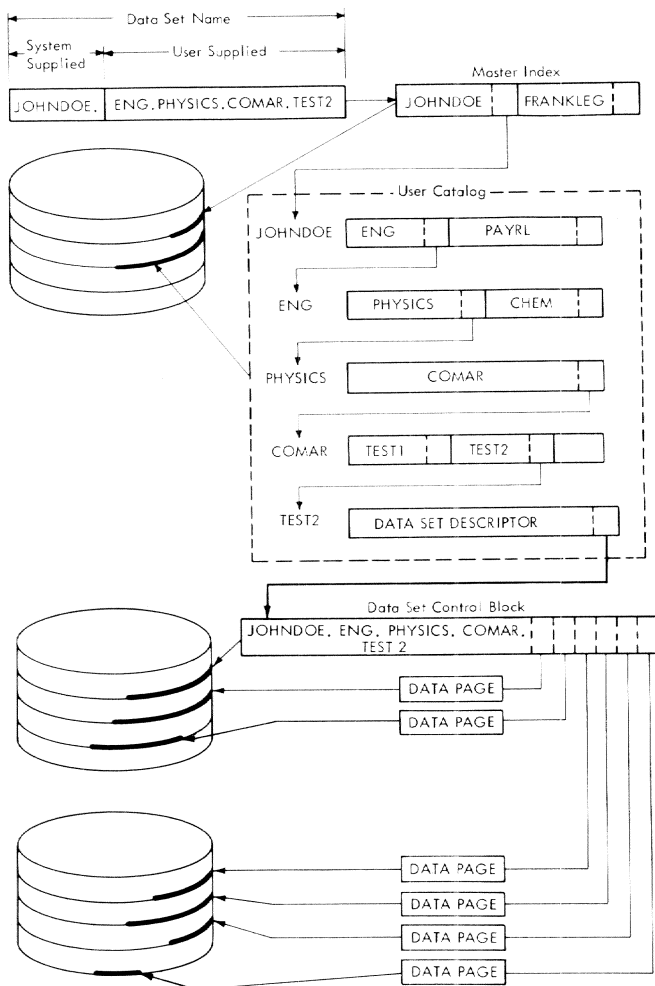


Figure 4. Catalog Example

1. You want to remove the catalog entry of a data set from the catalog without erasing the data set, and the data set resides on a private volume.
2. You want to remove the catalog entry of a data set you are sharing from your catalog (because you no longer have a need to share that data set).

The **ERASE** command can also be used for uncataloging. **ERASE** removes the catalog entry, and erases the data set as well *if* it resides on a direct-access volume. (Erasing means making the storage space of the data set available for other use.)

So that you can specify whether you want to be given one data set name at a time when you enter either the **ERASE** or **DELETE** command, provision is made to set the value of **DEPROMPT** (a value contained in your User Profile) to either **YES** or **NO**. If the value was set to **YES**, and you specify a partially qualified data set name, you will be given one data set name at a time for disposition. If the value was set to **NO**, all data sets grouped under this partially qualified name will be erased or deleted without individual presenta-

tion. If you specify a fully qualified name, the data set will be erased or deleted no matter what was specified for **DEPROMPT**.

You have the option in certain commands, as **PRINT** and **PUNCH**, if a cataloged data set is involved, of specifying whether it is to be erased or not after the output operation.

Using System Storage

The system assumes that you want storage on a public volume unless you specifically ask for storage on a private volume. When it is necessary to retain the data sets in the system, you make the most effective use of rss/360 by storing your data sets on public volumes. Public volumes are always mounted and available for allocation to your task, within the limits of public allocation established for you by your installation.

If you use private volumes, you may need to wait for devices to become available; in any case, you must wait for the operator to mount the volume on the device. Each time a request is made for a device on which to mount a private volume, the system must determine whether or not it can honor the request, based on the current requirements throughout the system for those devices. If the system cannot allocate a private device to your task, one of two actions occurs, depending upon the operational mode:

- In a conversational task, if the system cannot allocate the required space or if the required volumes cannot be mounted, the system issues a diagnostic message to you during the execution of the **DDEF** command. The system cancels the **DDEF** command, returns control to the terminal, and awaits another command.
- In a nonconversational task, the system places your task in abeyance until all private devices required by the task are allocated. You must include a **SECURE** command to reserve all devices that will be required for private volumes during the execution of a nonconversational task. **SECURE** must appear immediately after the **LOGON** command, and only one **SECURE** command is allowed for each task. The devices specified for private volumes will be reserved so that the task can be executed without waiting for I/O devices; any waiting that may be necessary to reserve the devices occurs at **SECURE** time rather than during execution time. The **SECURE** command is never used in a conversational task; it is mandatory only in nonconversational tasks that include references to private volumes.

Protecting and Sharing Data Sets

You cannot gain access to any data sets other than your own unless you have system authorization to do

so, or you have been given authorization to share them by another user who owns the data set(s) involved.

A *shared* data set is one that is cataloged and for which the owner has issued a `PERMIT` command. It belongs to one user, but may be shared with other users on any of the following bases:

1. *Read-only access*: The sharer may read the data set, but may not change it in any way.
2. *Read-and-write access*: The sharer can both read and write to the data set, but he may not erase it.
3. *Unlimited access*: The sharer, in effect, can treat the data set as his own; he may thus even erase it.

You issue a `PERMIT` command to designate other users who may share your data sets, and to indicate the level of access those users may have. You also use the `PERMIT` command to withdraw from previously authorized sharers the right to continue sharing your data. Each time you issue a `PERMIT` command, your catalog is updated when the task is terminated by `LOGOFF` or `ABEND`. Information on who can share which of your data sets is stored in your catalog.

If you have been named in another user's `PERMIT` command, you must issue a `SHARE` command before you can actually access the data sets he has authorized you to use. To see how this command is used, assume that a sharer's user identification is `JMC200` and that he has been permitted to share one data set. The data set is owned by user `RKP100`, and is cataloged by him under the fully qualified name `ENG.PHYSICS.COMAR.TEST2`. Assume also that the sharer wants to name the data set `ENG.CHEM.NOTAR.TEST1`. He would then issue the `SHARE` command shown at the top of Figure 5. In response to that command, the system would search the owner's catalog to see if the prospective sharer is authorized. If he is not, the command is ignored; if he is authorized, the system places the owner's complete name for the data set in the sharer's catalog with a pointer back to the master index. Whenever the sharer subsequently refers to the data set by his name, the system locates the data set by the search procedure shown on Figure 5.

To be concurrently accessible by more than one task, a data set must be cataloged and must be a virtual storage data set (`vs`, `vi`, or `vp`).

Maintaining Program Libraries

A program in `TSS/360` can consist of one or more object modules. An object module is the output of a language processor or the linkage editor (exclusive of the listing). All programs in `TSS/360` are stored in object module form in program libraries, which are `vp` data sets. A program consisting of only one object module is stored entirely within one library; however, if a program consists of several object modules, those

modules may reside in different libraries, depending on how you store them.

There are four categories of program libraries:

- System library (`SYSLIB`)
- User library (`USERLIB`)
- User-defined job libraries
- Linkage editor libraries

`SYSLIB` is accessible to all users on a read-only basis.

`USERLIB` is the private library assigned to you when you are joined to the system. This library is automatically built for you and made a part of your catalog by the system. `USERLIB` is thus available each time you log on. If you do not use job libraries in a task, all the object modules resulting from your use of the language processors are automatically placed in `USERLIB`. You may wish to restrict your `USERLIB` to object modules that you execute frequently or that you use frequently in the buildup of other object modules.

The program library list is a defined hierarchy of program libraries. It is initialized at log-on time, and at that time consists of `USERLIB` and `SYSLIB`. The library at the top of the list always automatically receives all object modules resulting from language processing. As noted above, if no job libraries are defined, the library at the top of the list is always `USERLIB`. However, you can specify that a job library be added to the program library list to receive the output of the language processors. You do this by issuing a `DDEF` command defining that job library and containing the `OPTION=JOBLIB` parameter. When this command is executed, the name of that job library is added to the top of the program library list. That library then receives all subsequent output of the language processors until another job library is defined (and it is placed at the top of the list), or until a `RELEASE` command is issued for the first job library. The `POD?` command can be used to obtain a list of member names, alias (entry point) names, and other member oriented data from the task's `USERLIB` and such cataloged job libraries as have been established by the user.

In addition to using the program library list to store object modules, the system also uses this list to control its order of search when looking for object modules that must be loaded at execution time. The library at the top of the list is always searched first, then the next-to-the-top library, etc.; then, `USERLIB` and, finally, `SYSLIB`. By using the linkage editor, you can move object modules from one library to another.

Other user-defined libraries may be defined by `DDEF` commands that omit the `JOBLIB` parameter. They are not placed on the program library list and cannot be loaded. They are used principally with the linkage

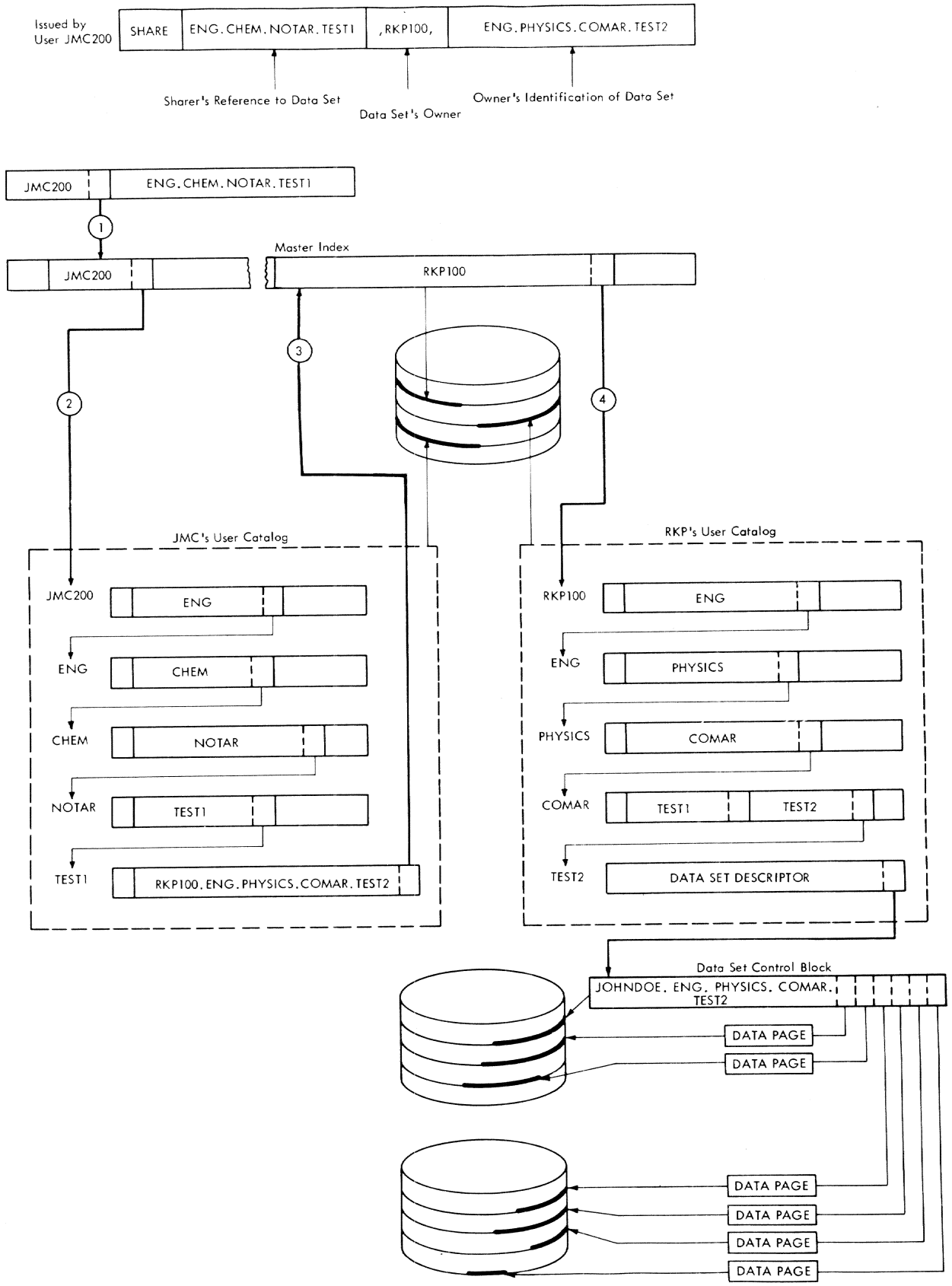


Figure 5. Sharing of Cataloged Data Sets

editor. (Refer to *Linkage Editor* for an explanation of linkage editor library control.)

Copying, Modifying, and Erasing Data Sets

You can use the `CDS` command to make a copy of any data set (or any member of a partitioned data set) to which you have access except data sets whose records are in undefined format (see Appendix E), such as program module libraries. You can also use the command to renumber the lines of a line data set as it is being copied. Both the original and copy data sets must be defined in your task.

You can use the `MODIFY` command to insert, delete, replace, or inspect records of a `VI` data set, or of a `VI` member of a `VP` data set. You have to identify the record to be modified (by its key or line number). You can review modifications, and play back corrected lines for confirmation of your changes.

You can use the `VV`, `VT`, and `TV` commands to copy your data sets, depending on their origin and desired destination. The `VV` command causes a `VAM` data set to be copied into public storage. The `VT` command causes a `VAM` data set to be reproduced on 9-track magnetic tape. The `TV` command retrieves and writes into public storage a data set previously written on 9-track magnetic tape by the `VT` command.

You can use the `ERASE` command to erase data sets that you own. If you are sharing someone else's data set, you can remove its entry from your catalog by issuing the `DELETE` command.

Conversational Task Termination

To terminate your conversational task, issue a `LOGOFF` command. The system will then update its internal accounting tables reflecting your use of the system during the session.

If you later want to communicate with the system again conversationally, you must again log on as described in the section "Conversational Task Initiation."

Nonconversational Use of the System

There are many applications where you will not require dynamic communication with the system or with your problem programs in order to obtain the problem solutions you desire.

Nonconversational Task Initiation

Figure 6 illustrates the various ways in which you can use the system for nonconversational processing.

You can issue the `EXECUTE` command in a conversational task to initiate nonconversational tasks. The `EXECUTE` command names a cataloged command procedure that is to be executed. The command procedure functions as the `SYSIN` data set for the nonconversational task. It must begin with a `LOGON` command; end with a `LOGOFF` command; and, it must be pre-stored in the system by you so that it can be retrieved merely by its name. If private devices are required in the task, a `SECURE` command must immediately follow the `LOGON` command.

You can issue `PRINT`, `PUNCH`, and `WT` commands in either a conversational or nonconversational task. These commands are, in effect, one-command-procedures. They initiate nonconversational tasks that transfer data between a direct-access device and a printer, card punch, or tape unit, respectively.

You can also have the operator initiate nonconversational tasks for you. You supply him with a card deck or magnetic tape; the contents of the deck or tape depend on what you want done:

- If you want to enter data into the system for later use (i.e., prestore it) you prepare a card deck (or magnetic tape) with a command procedure of the following form:

```

DATASET descriptor card
Data cards
%ENDDS card
    } card images
    
```

If you do this, the task set up by the operator will transfer data from the input medium to a direct-ac-

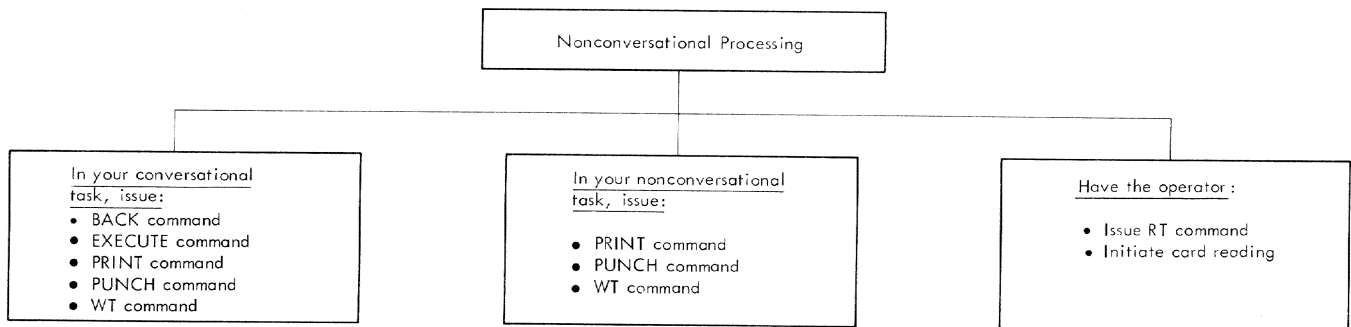


Figure 6. Nonconversational Task Initiation

cess device and catalog it so that it is later available to you by its name.

- If you want to enter a command procedure, you prepare a card deck as follows:

LOGON card
Other commands & data cards
LOGOFF card

The task that is set up by the operator will execute the commands in the command procedure you have defined.

In all of the ways in which a nonconversational task is initiated, the system action is basically the same:

1. The request to set up the nonconversational task is enqueued and assigned a batch sequence number.
2. The individual requesting the task (you or the operator) is sent the batch sequence number (to later permit that individual to CANCEL that task if he wants).
3. The requested task is then executed when the required system resources become available.

Nonconversational Command Procedure Processing

When you use the EXECUTE command to initiate a nonconversational task, the commands are taken one at a time from the cataloged command procedure (SYSIN data set) you specified. The system specifies the task's SYSOUT. You can read SYSIN input in your programs, in a manner similar to conversational mode, if the data is properly positioned in the SYSIN data set. Similarly, you can write to SYSOUT from your program. Because there is no prompting in nonconversational processing, you must specify every command completely, you must take care to have your commands in proper sequence, you must include a SECURE command to obtain any devices needed for private volumes, and you must catalog any data sets you want to keep.

In nonconversational mode, listings produced by language processors (by the FORTRAN compiler, the assembler, or the linkage editor) are written automatically on SYSOUT unless you specify the LISTDS option (in the FTN, ASM, or LNK command) as "Y". If the LISTDS option is "Y", the listing is put into the list data set, as in the conversational mode, and will not be printed until you issue a PRINT command.

Nonconversational Task Termination

The execution of nonconversational tasks (except PRINT and PUNCH) is terminated when their LOGOFF

command is executed. The system then automatically prints out the task's SYSOUT data set. For nonconversational tasks, the SYSOUT data set consists of the commands from SYSIN that were executed, any data that your program writes to SYSOUT, and the compiler-issued diagnostic messages (if no listings were requested).

Tasks created by the PRINT and PUNCH commands terminate when the data transfer is completed.

You can also terminate your nonconversational tasks by issuing a CANCEL command identifying each task to be terminated by its batch sequence number.

Mixed Mode Use of the System

You can begin a task at your terminal, and then issue a BACK command to have the task's execution completed in the nonconversational mode. Before issuing the BACK command, you must have stored a SYSIN data set that is to function as the command procedure and, if desired, input data for the nonconversational portion of your task. You must also have issued DDEF commands for any private volumes you may need. The SYSIN data set must not contain a LOGON command (because you have already logged on), but it should end with a LOGOFF command.

When you issue a BACK command for a task, the system determines whether it can provide sufficient resources to continue your task nonconversationally. If it cannot, the system will reject your request, and you can try later.

Once your BACK request is accepted, your terminal is inactive. You must then log on at your terminal again to initiate a new conversational task if you want to continue to use the terminal.

Remote Job Entry

The TSS/360 remote job entry feature makes high-speed printing and card reading available at locations outside the central computer installation. Each remote station has a card reader to accept input and a printer to produce output.

A complete description of this facility is provided in *IBM System/360 Time Sharing System: Remote Job Entry, GC28-2057*.

Command Directory

Table 1 presents a guide to the commands of TSS/360 as presented in the examples and appendixes of this book. The commands are grouped by general function.

For each command, a sample usage and a statement of its general effect are shown, along with the numbers of all the examples in Part II in which this particular command appears.

Table 1. Command Directory

FUNCTION	COMMAND	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
Task Management	BACK	BACK DSNAME=PROC12A	Switches your conversational task to non-conversational mode. Here you specify PROC12A as the source of further commands.	12
	CA CB	CA	Specifies that card input will follow. Causes SYSIN to be switched to the card reader. The A signifies that you want to convert card input from 1057 card punch code to EBCDIC. A CB would signify conversion from 029 keypunch code to EBCDIC.	20
	CANCEL	CANCEL BSN=0375	Terminates execution of nonconversational task to which the system had assigned batch sequence number 0375.	19, 20
	EXECUTE	EXECUTE DSNAME=PROC12	Requests the execution in nonconversational mode of a sequence of commands contained in data set PROC12. You may then continue in conversational mode at the terminal.	12
	KA KB	KB	Specifies that keyboard input in folded mode will follow (i.e. the lower case characters a-z and ! " # \$ % & ' () * + , - . / : ; < = > ? [\] ^ _ ` { } ~ are to be translated into their upper-case equivalents (A-Z and \$ # @). KA would specify that you want to use the full EBCDIC character set.	20
	LOGOFF	LOGOFF	Notifies system that you want to terminate your task. The system may then query you regarding any uncataloged data sets.	All but example 3
	LOGON	ADUSERID, MYPASS*, ADACCT29	Identifies you to the system. You enter your identification, password, and account number.	All
	PROCDEF	PROCDEF ZLOGON	Defines a procedure (ZLOGON) which is automatically executed each time you log on, prior to any terminal processing you may initiate.	27
	SECURE	SECURE (TA=1,9)	Reserves devices for private volumes required for nonconversational tasks. This command at the beginning of your sequence of commands secures one 9-track tape unit.	14, 21
	TIME	TIME 15	Establishes the maximum amount of elapsed time that a task will be allowed to run. Here you have set a 15-minute limit on program execution time.	2
USAGE	USAGE	Presents totals of system resources used since LOGON and since you were joined to system.	22	
Data Set Management	CATALOG	CATALOG DSNAME= RESULT01, STATE=N, ACC=R	Causes your physical sequential output data set RESULT01 to be cataloged. The N indicates that the catalog entry is new. The R indicates that you want to restrict access to the data set to a read-only.	5, 11, 21, 23, 24
	CDD	CDD DDPACK, DDMAIN14	Causes execution of the DDEF commands defined by the data definition name DDMAIN14, which you had stored in the data set DDPACK.	14
	CDS	CDS DSNAME1=MYDATA, DSNAME2=MYDATA1	Copies the data set MYDATA, naming the copy MYDATA1.	18

Table 1. Command Directory (cont.)

FUNCTION	COMMAND	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
Data Set Management (Continued)	CLOSE	CLOSE	Closes data sets from the command level when normal processing has been interrupted and closure from the program level is difficult or impossible.	17
	DDEF	DDEF DDNAME=LIBDD, DSORG=VP, DSNAME=SCRATCH, OPTION=JOBLIB	Defines a data set for the current task. Every data set you use must be defined for the current task, even if previously cataloged. You assign LIBDD as the name of the data definition. The data set created by this DDEF is a virtual partitioned job library named SCRATCH.	4, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 21, 23, 24
	DELETE	DELETE DSNAME=DATA5	Removes the entry for the data set DATA5 from your catalog.	22
	DSS?	DSS? PHI,SIGMA	Causes printing of information about your data sets PHI and SIGMA.	22
	ERASE	ERASE DSNAME=SCRATCH	Erases data set SCRATCH (releases direct-access storage for other use), and, if cataloged, deletes name from catalog.	4, 5, 7, 8, 19, 22
	EVV	EVV DEVICE=2311, VOLUME=(123ABC), USERID=ADUSERID	Presents a private VAM volume to the system and catalogs the data sets on it. The volume is a 2311 with volume serial number 123ABC. The data sets are to be cataloged in user ADUSERID's user catalog.	
	PC?	PC? PHI,SIGMA	Causes printing of limited information about your data sets PHI and SIGMA.	22
	PERMIT	PERMIT DSNAME=DATA, STATE=N, ACCESS=RO, USERID=ABPALID	Permits the user with the user ID,ABPALID, to have read-only access to all of your data set whose left-most name qualifier is DATA. The N signifies that this command creates a new sharer's list, rather than updates on existing list.	18
	POD?	POD? PODNAME=USERLIB	Causes printout of information about each object module in your USERLIB.	22
	RELEASE	RELEASE DDNAME=MYDD	Revokes the data definition established by a previously issued DDEF named MYDD.	10, 17
	SHARE	SHARE DSNAME=MYDATA, USERID=ADUSERID, OWNERDS=DATA	Creates an entry in your catalog for the data set named DATA, to which you are authorized access when the owner issues a PERMIT command.	18
	RET	RET DSNAME=ALPHA, RET=(TCU)	Modifies the RET field of the data set descriptor to specify that data set ALPHA be assigned to temporary storage with read/write access and that it be deleted when it is closed.	16
	TV	TV DSNAME1=COPY1, DSNAME2=COPY2	Causes data set copied on 9-track as COPY1 to be reproduced on direct-access storage in VAM format under the name COPY2.	24
	VT	VT DSNAME1=ORIGIN1, DSNAME2=COPY1	Causes VAM data set named ORIGIN1 to be copied on 9-track magnetic tape as COPY1.	24
VV	VV DSNAME1=COPY2, DSNAME2=COPY3	Causes VAM data set named COPY2 to be copied into public storage under the name COPY3.	24	

Table 1. Command Directory (cont.)

FUNCTION	COMMAND	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
Text Editing	CORRECT	CORRECT N1=100, SCOL=8	Causes the characters beginning with column 8 of line 100 in the current region to be displayed for correction purposes. Characters are inserted, deleted or changed by keying in correction symbols directly beneath the affected character(s).	26
	DISABLE and ENABLE	DISABLE ENABLE	DISABLE causes your modifications to the data set to be provisional, pending execution of an ENABLE command. By issuing a STET command instead of an ENABLE, you would cause all modifications made after the DISABLE to be deleted.	26
	EDIT	EDIT DSNAME=EX26	Invokes the facilities of the Text Editor. The operand identifies the data definition name associated with the data set which is to be edited.	25, 26
	END	END	Terminates processing of the Text Editor or the PROCDEF command.	25, 26
	EXCERPT	EXCERPT DDNAME=NEW1, RNAME=REGION2, N1=600,N2=1000	Excerpts lines 600 to 1000 from REGION2 of the data set associated with the data definition name NEW1 and inserts them into your current region. EXCERPT, following a REVISE command, replaces a range of lines in the current data set; following an INSERT command, it adds to lines being typed in from the terminal.	26
	EXCISE	EXCISE N1=0000200	Deletes line number 0000200 from the current region.	25
	INSERT	INSERT 0000400	Informs the system that you wish to insert the following lines from SYSIN into the current region, placing them immediately after line 0000400.	25
	LIST	LIST N1=100, N2=500	Lines 100 to 500 of the current region are displayed.	26
	LOCATE	LOCATE STRING=LINEF	Searches the current region for the first occurrence of character string LINEF. When the string is found, the line containing it is displayed.	26
	NUMBER	NUMBER N1=300, N2=500, BASE=300, INCR=50	Causes the lines within the current region to be renumbered. N1 and N2 define the range to be renumbered. The numbering will begin with 300 and increase in increments of 50.	26
	POST	POST	Makes all previous editing changes permanent and puts Text Editor into disabled state, making future changes reversible.	26
REGION	REGION RNAME=FIXA	Used after invoking the Text Editor, this command identifies a region name to be assigned a line or range of lines that are to be edited.	25, 26	
Data Editing	DATA	DATA DSNAME=PROC12A	Creates a new data set named PROC12A. You do not need to issue a DDEF command for data sets created by a DATA command. By default, the data set organization is VS.	12, 14, 16
	LINEP	LINEP SOURCE.MAIN9, (1200,1400)	Causes printing of source lines 1200 to 1400 from the specified line data set.	9, 10, 16, 19
	MODIFY	MODIFY SETNAME= SOURCE.MAIN4	Permits you, with subsequent parameters, to insert, replace, review, or delete lines in a VISAM data set named SOURCE.MAIN4 containing FORTRAN source statements for your program MAIN4.	5

Table 1. Command Directory (cont.)

FUNCTION	COMMAND	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
Bulk Output	PRINT	PRINT DSNAME= LIST.MAIN2(0), ERASE=Y, PRTSP=EDIT	Causes the current generation of the listing data set LIST.MAIN2(0) to be printed on the high-speed printer.	2, 4, 5, 6, 8, 10, 13, 14, 19, 20
	PUNCH	PUNCH DSNAME= SOURCE.MAIN19, STARTNO=9, ENDNO=88	Causes the 9th through the 88th characters of each record of your source data set to be punched. The first eight characters are the line number and input key. Punching is done in nonconversational mode when system resources are available.	19
	WT	WT DSNAME=M22OUT	Causes the data set M22OUT to be written on magnetic tape for subsequent off-line printing.	22
Program Management	FTN	FTN NAME=MAIN2	Activates the FORTRAN compiler. You name the object module to be produced in this compilation MAIN2. The system will assign the name SOURCE.MAIN2 to the data set it creates containing your FORTRAN source statements. You have specified the default values for the remaining parameters by omitting them.	2, 3, 4, 5, 8, 13, 20
Program Control	AT	AT MAIN.17, MAIN.96(2)	Causes message to be printed on SYSOUT when execution of your program MAIN reaches statement numbered 17 and when it reaches the first executable statement after the one numbered 96.	10
	CALL	CALL MAIN4	Causes loading and execution of your program MAIN4 at the first executable statement.	4, 5, 7, 9, 11, 13, 14, 15, 17, 20, 21, 23
	DUMP	DUMP MAIN10#C, MAIN10#P	Causes a formatted dump of MAIN10's CSECT and PSECT. It will be written in the PCSOUT data set for later printing.	10
	GO	GO	Causes execution of your current program to be resumed from the point of interruption.	9, 10, 15
	IF	AT 101; IF A>B; STOP	Causes execution of your program to stop at location 101 if the expression A>B is true.	10
	DISPLAY	DISPLAY MAIN9.ALPHA, MAIN9.BETA	Causes printout on SYSOUT of the current contents of data fields ALPHA and BETA in program MAIN9.	9
	LOAD	LOAD BLKDATA8	Causes your block data subprogram to be transferred from your library to virtual storage.	8, 10
	QUALIFY	QUALIFY MNAME=MAIN9	Causes subsequent references to symbols and statement numbers to be qualified by the name MAIN9. You can now write AB instead of MAIN9.AB, or 105 instead of MAIN9.105.	9, 10
	REMOVE	REMOVE 2	Deletes previously issued AT commands, or PCS statements containing AT commands. Here you specify deletion of the statement to which the system assigned the number 2.	10
	SET	SET BETA=1.0	Sets value of BETA equal to 1.0.	9, 10
	STOP	STOP	Stops program execution and causes printing on SYSOUT of current instruction location and program status information.	9
UNLOAD	UNLOAD MAIN15	Removes the object module MAIN15 from your virtual storage.	15, 21	

Table 1. Command Directory (cont.)

FUNCTION	COMMAND	SAMPLE USAGES	EFFECT	ILLUSTRATIVE EXAMPLES
User Profile Management	DEFAULT	DEFAULT DSORG=VS	Sets the default value for the DSORG parameter of the DDEF command to virtual sequential (VS).	28
	PROFILE	PROFILE	Causes changes affected by DEFAULT and SYNONYM commands to become a permanent part of your user profile.	28
	SYNONYM	SYNONYM DOPROG=FTNPGM	Establishes a synonym for the name FTNPGM. This module can now be called by the name DOPROG.	28

Part II is devoted to examples in which the dialog between you and the system appears (along with explanatory comments) as it would at the terminal. They are typical examples of system use. Unlike the sample programs in Appendix I, the examples in this section have not been system-tested. You may, therefore, observe minor differences between an example's description in Part II and the printout you obtain if you run the example itself. Use the examples, therefore, only as a learning device, and as models for designing your own work.

Commands and concepts are presented in an ordered sequence: the most necessary and basic ones appear first, and are reviewed in subsequent examples. The examples are designed so that the beginner should read them in sequence. Those familiar with the commands and concepts can use the examples for reference.

FORTRAN programs are shown where they are necessary to clarify use of the commands. Only the relevant statements are included.

The system issues various types of messages at your terminal, as follows:

- Prompting Messages — request that you supply command operands or other information. You are prompted only for omitted parameters that have no default option.
- Response Messages — either inform you of actions the system has taken in executing a command or request additional information.
- Diagnostic Messages — inform you of errors and prompt you for correction.

In these examples, lines typed by the system are headed `sys`, lines you enter are headed `you`. Lines in which both the system and you enter something are headed `s,y`. Lines printed by your program are headed `pgm`, and cards entered from the terminal card reader and printed are headed `crp`, for "card image printout."

Example 1: Initiating and Terminating a Conversational Task

In this example, you initiate a simple conversational task and then terminate it. The commentary explains the keyboard entries required to converse with the system.

To begin a conversational task, make sure that the terminal is properly prepared (refer to instructions provided by your installation or to the *Terminal User's Guide*). When you dial up the system or press the attention button for the first time in your task, the system assumes a log-on operation and responds with the current date and time. You then enter all the log-on operands.

During your dialog with the system, your commands are not entered into the system until you press the return key.

YOU: (press the attention button or dial up system)

From this point on, pressing the attention button halts current activity in most situations. Consult Appendix F for the specific action taken in each situation.

YOU: LOGON ADUSERID,MYPASS*, ,ADACCT24,A
9,A,,P

You must enter the entire LOGON command on a single line. While typing the LOGON operands, you realize that you have entered your charge number incorrectly. Therefore, you backspace three characters, move the paper up one line by hand to avoid overtyping, and reenter the corrected portion of the charge number. You then complete the LOGON operands. If you wanted, you could have cancelled the entire line by typing a pound sign (#) and immediately pressing the return key; then you would reenter the correct line.

SYS: (responds with the current release-level of the system, the date, time, and task identification)

You can now communicate with the system by entering commands.

Explanation of LOGON Operands

ADUSERID

First Operand — User Identification

This operand is the full identification assigned to you when you were joined to the system.

MYPASS*

Second Operand — Password

This operand is an installation-assigned code that provides protection against unauthorized use of your user identification. In conversational mode, you must supply a password if one has been assigned to you.

Third Operand — Addressing

Specifies whether 24-bit or 32-bit addressing is to be used for this task. If you default this operand, the installation default value will take effect.

ADACCT29

Fourth Operand — Charge Number

This operand is the charge or account number that was assigned to you by your administrator. The first two characters of your charge number also identify your administrator.

A

Fifth Operand — Control Section Packing

This operand specifies whether control sections are to be packed (i.e., not placed on separate pages), and the manner of packing to be used. The codes and their meanings are:

Code	Meaning
A	Pack all control sections
P	Pack all prototype control sections (PSECTs)
O	Pack all control sections having neither public nor prototype attributes
X	Pack all control sections except prototype control sections
N	No packing

Sixth Operand – Maximum Auxiliary Storage

This operand specifies the maximum amount of auxiliary storage to be allocated to your task; you default this operand and use the installation default value.

P

Seventh Operand – Pristine Mode

This operand allows you to log on with only the system-supplied defaults, synonyms, procdefs, and Character and Switch Table. Since you specified this operand as P, your user library is defined; if you had specified it as X, your user library would not be defined.

S, Y KB

After logging you on, the system prints a single underscore and then backspaces; this is the standard signal that it is ready to receive your next command on the same line. Here you specify that you want folded mode; that is that certain lower case characters (as a-z and ! “e) be translated by the system into their upper case equivalents (A-Z and \$ # @, respectively). Thus, with KB, you no longer need to perform shifting operations.

When you initiate a conversational task, the system automatically assumes folded mode; hence in this example you need not have specified KB. However, there are other character control commands, such as KA, which invoke EBCDIC mode at the keyboard. Thus, if you specify KA and at a later time in your session wish to return to folded mode, you must enter KB.

Here you decide to conclude your session, so you logoff. Note that LOGoff translates to LOGOFF.

YOU: LOGoff
SYS:

The system confirms your LOGOFF command.

Example 2: Compilation and Correction from the Terminal

In this example, you type in the source statements of a short program and correct several errors while compiling the program. The compiled object module is stored in your USERLIB. The listings you select are printed as a separate task, only if requested using the PRINT command. After logging on, you issue:

S,Y: TIME 15

The TIME command establishes a period of time a task will be allowed to run in virtual storage. You decide that 15 minutes will be adequate for your task and wish to be alerted when this interval is exhausted. TIME is thus useful in controlling inadvertent loops and other abnormal actions occurring in programs.

S,Y: FTN NAME=MAIN2

This command activates the FORTRAN compiler. A compiled program is called an object module. You name the object module to be produced in this compilation MAIN2. The system creates a source data set, naming it SOURCE.MAIN2, which will contain your FORTRAN source statements as you entered at the terminal.

Because SOURCE.MAIN2 is a line data set residing in public storage, it is automatically cataloged for you. Although you did not explicitly issue a DDEF command, there was an implied system issuance of DDEF associated with your FTN command.

The system also creates a list data set, named LIST.MAIN2, which will contain the listing of the object module. The NAME parameter is not defaultable, although you could enter the command as FTN MAIN2. It is permissible to omit the NAME keyword. Note, however, that when you omit the keyword and have several operands to enter, you must specify the operands in the order in which they are syntactically defined. In this example you specify the default values for the remaining parameters by simply omitting them.

S,Y: 0000100 READ (5,10) A, B

S,Y: 0000200 FORMAT (F6.2)

The system prints line numbers. After each line number, you skip a space and enter a FORTRAN statement. You do not have to follow the FORTRAN card format when entering source lines from the terminal. Skipping one space after the system prints the line number improves readability. Since the system regards any line with a C in column 1 as a comment line, skipping a space also prevents lines such as C = A*B from being treated as a comment. When you want a comment line, you should not skip a space, but enter the C in column 1.

SYS: 0000200 E *** FORMAT STATEMENT DOES NOT HAVE STATEMENT NUMBER.

The compiler examines each statement for syntactical errors as soon as it is received. 200 is the line number of the statement in error, E is an error level code, indicating a serious program error; the statement is ignored. Other error level codes are:

W A warning of a possible problem; the statement is compiled as written.

F Serious error; statement can only be partially compiled.

A Compilation cannot be continued.

See Appendix A for further details on compiler diagnostics.

SYS: 0000200 FORMAT (F6.2)

The system prints the line in error for your review.

S,Y: #200,10 FORMAT (F6.2)

The system prints the number sign, #, after which you enter the line number of the line to be corrected, followed by a comma and the replacement line. There must be at least one space between the statement number (10) and the rest of the statement.

S,Y: #(press return key)

Another correction or change or even insertion of an entirely new line could be made at this time. Since you wish to continue entering your program, you request the next line number by pressing the return key.

S,Y: 0000300 ATB = A * B
S,Y: 0000400 WRITE (6,20) A, B, ATB
S,Y: 0000500 STOP
S,Y: 0000600 END

SYS: 0000400 F *** 20 STATEMENT NUMBER USED AS FORMAT IS NOT DEFINED.

After the END statement has been entered, the compiler diagnoses global errors, which are errors that involve more than a single statement. In doing so, it found an error.

SYS:

The system invites you to modify your source statements.

YOU: Y

You may reply yes (Y) or no (N). Here you reply yes and wait for the system to invite your modification.

S,Y: #400, WRITE (6,10) A, B, ATB

S,Y: #(press return key)

While you are entering modifications, no error checking is done; however, the compiler rescans the entire program when it recompiles after modifications are completed.

SYS:

The system continues compilation and informs you when finished.

S,Y: PRINT LIST.MAIN2(0), ERASE=Y, PRTSP=EDIT

The system will establish a nonconversational task to print the current generation of LIST.MAIN2. The current generation is specified by the (0) immediately after the generation data group name. The PRTSP operand is specified as EDIT because the system supplies control characters to format the listing. The ERASE operand is included to eliminate the listing from the system after printing. Each listing is put into the list data set (the data set beginning with LIST.).

SYS:

The system acknowledges your PRINT command and informs you of the batch sequence number it has assigned to the printing task.

Now compilation is complete.

You are informed of the batch sequence number of the separate task created to print the listings produced by the compiler.

The compiled object module now resides on the library at the top of your program library list — in this case, your USERLIB.

S,Y: LOGOFF

SYS:

The system will confirm your LOGOFF command.

Example 3: Compilation and Correction from the Terminal

In this example, you type in the same FORTRAN program as in Example 2, but this time all applicable parameters are shown. After logging on, you issue:

```
S,Y: FTN NAME=MAIN3,STORED=N,VERID=A6/26,-  
     ISD=N,SLIST=Y,OBLIST=N,CRLIST=N,STEDIT=N,MMAP=N,-  
     BCD=N,PUBLIC=N,LISTDS=N,LINCR=(100,100)
```

This is the same FTN command as in Example 2, except that here all the FTN operands are entered with their keywords. Each operand is described below. When entering commands conversationally at the terminal, you may continue them on another line if necessary by entering a hyphen at the point where you wish to break the current line.

Explanation of FTN Operands

- NAME=** **Object Module Name**
You assign the name MAIN3 to the output object module to be created by the compiler. The source data set for this object module is named SOURCE.MAIN3; the listing data set is named LIST.MAIN3(0).
This is the only FTN operand that may not be defaulted.
- STORED=** **Prestored Source Data Set**
You specify N so that you may enter source statements from your terminal rather than compile from a prestored source data set.
- VERID=** **Object Module Version Identification**
You may assign a version identification to the object module in this case, A6/26. It appears on output listings of the named program and is stored in a special field in your object module. If you do not assign a version identification, you may distinguish the version of your object module by using the system supplied "time stamp." A time stamp is always produced by the system; it gives the current time and date at which compilation begins.
- ISD=** **Internal Symbol Dictionary**
This parameter permits you to create an Internal Symbol Dictionary (ISD) during compilation. An ISD is necessary for the fullest use of the program control system (PCS).
- SLIST=** **Source Listing**
The listings you request with this and the next four parameters will form your listing data set. (See Appendix A for a detailed explanation of these listings.) The system creates a name for this data set by prefixing "LIST." to the module name you supplied as the first assembler parameter (LIST.MAIN3), using generation data group logic.
The source listing shows the source input statements.
- OBLIST=** **Object Listing**
You indicate you do not want an object listing. The object listing shows the code generated by the compiler. This code is in the form of assembler language statements and hexadecimal machine language code. Ordinarily, it is not needed by FORTRAN programmers.
- CRLIST=** **Cross Reference Listing**
This listing shows where, in the source program, each statement number and symbol is defined or referred to. You have told the system not to produce this listing.
- STEDIT=** **Edited Symbol Table**
This listing indicates the characteristics and displacement for every symbol in the source program. You have indicated you do not want a listing of this table.

MMAP=

Memory Map

This listing contains summary information about the module, most of which is included on other listings. You have declined a listing of a memory map.

BCD=

Binary Coded Decimal

Since you are writing this at the terminal, the characters entered will be stored internally in the standard system EBCDIC form. If your source program were in the form of cards or tapes from a system using BCD, the BCD code could be used. In BCD mode, characters of either EBCDIC or BCD can be entered.

PUBLIC=

Private or Public CSECT Attribute

This parameter specifies whether the executable portion (not variables) of the object module is to have a public or private attribute. You have selected a private attribute. Public means that the resulting object program can be shared by other users. Most FORTRAN users will want to take the default option (i.e., private). For more detailed information regarding all FORTRAN compiler options, see Appendix A.

LISTDS=

Listing Destination

This parameter specifies whether the listings you request from the compiler are to be placed in a list data set or placed directly on SYSOUT; it is ignored in a conversational task.

LINCR=

Starting Line Number, Increment

The system creates a line number for each of your source statements. Although a part of the source data set being formed (SOURCE.MAIN3), line numbers are not an intrinsic part of the FORTRAN program itself and have no specific relationship to any statement numbers.

The starting line number and the increment number may contain three to seven digits, of which the last two must be 00. Thus in the case here illustrated, the system generated line numbers will be 100, 200, 300, etc.

SYS:

The system requests each input source line by typing out a line number to the terminal starting with 100. After you enter a source line, it is added to the source data set and processed by the FORTRAN compiler.

```
S,Y: 0000100 READ (5, 10) A, B
S,Y: 0000200 FORMAT (F6.2)
SYS: 0000200 E *** FORMAT STATEMENT DOES NOT HAVE STATEMENT NUMBER.
SYS: 0000200 FORMAT (F6.2)
```

The system prints the line in error for your review.
After the system prints the #, you correct the erroneous statement.

```
S,Y: #200,10 FORMAT (F6.2)
S,Y: #(press return key)
```

... and indicate that you have finished making modifications.
The session continues as in Example 2.

Example 4: Compile and Run

In this example you enter and compile a short test program without error and then execute it. You execute your task in conversational mode, using your terminal for both input and output. After logging on, you issue:

S,Y: DDEF LIBDD,VP, SCRATCH, OPTION=JOBLIB

This command defines a job library and causes it to be placed at the top of your program library list. Object modules produced by compilations will be placed in it instead of in your USERLIB, which is now second on your library list.

A DDEF command defines a data set during the session in which the command appears. In general, every data set you use must be defined for the current session, even if it has been previously cataloged.

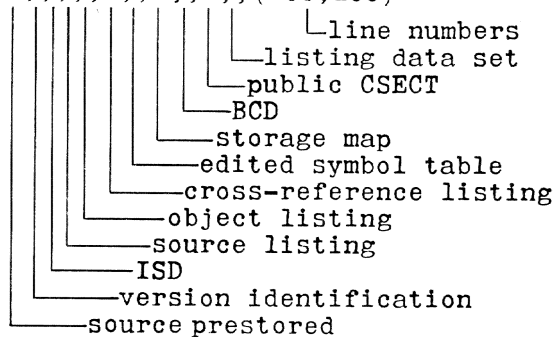
LIBDD is the data definition name (DDNAME) for the job library data set named SCRATCH. All libraries have virtual partitioned (VP) organization.

The OPTION indicates to the system that the data set you are defining is a job library.

The system automatically catalogs SCRATCH as a new catalog entry and assigns an "unlimited" (read/write) access qualifier to the library.

Your object module, MAIN4, will be placed automatically in your job library, SCRATCH, when compilation has satisfactorily completed.

S,Y: FTN MAIN4,N,,,,Y,,Y,,Y,,(100,100)



Because no keywords were used in specifying which operands you desired, you must enter those operands in the order shown.

Only the first operand must be specified. The rest may be defaulted by including a comma where the operand would appear. You must supply commas for defaults prior to the non-defaulted parameter (e.g., the Y for public CSECT attribute), but you need not supply trailing commas.

The system automatically catalogs SOURCE.MAIN4 as a new catalog entry and assigns an "unlimited" (read/write) access qualifier to the library.

```
S,Y: 0000100    READ (5,10)A
S,Y: 000020010  FORMAT (F6.2)
S,Y: 0000300    ATC = A*3.14
S,Y: 0000400    WRITE (6,10)A,ATC
S,Y: 0000500    STOP
S,Y: 0000600    END
```

You type in your program statements following the line numbers printed by the system. You use a tab stop to space over 6 columns so that the statements have the appearance of the standard card format. Such tabbing is not required, but it is recommended as a visual aid to avoid errors. When entering source lines, a tab has the logical effect of a single blank character. Example 2 shows a source program entered in free form. Appendix A discusses the use of tabs and spaces in entering source statements.

SYS:

The system informs you of the satisfactory completion of the compilation.

S,Y: Print LIST.MAIN4(0),PRTSP=EDIT
SYS:

The system acknowledges your PRINT command and informs you of the batch sequence number it has assigned to the printing task.

S,Y: Call MAIN4

The CALL command has two effects: it causes the loading of your object module and initiates program execution at the first executable statement. Note that the NAME keyword is omitted for the sake of brevity.

Since you gave no DDEF command for the data set reference number 5 in your READ statement, the system will assume you want to obtain input from SYSIN, which is the terminal when in conversational mode.

SYS: (unlocks terminal keyboard)

YOU: 0.5

PGM: 0.50

1.57

Since you gave no DDEF command for the data set reference number 6 in your WRITE statement, it will be defined as SYSOUT, which is also the terminal when in conversational mode.

PGM: TERMINATED: STOP

This message is printed when execution reaches your STOP statement (source line 500). You are now returned to command mode, indicated by the printout of the underscore.

S,Y: Erase SCRATCH

Before logging off you decide that you want to erase the job library data set, SCRATCH, containing the object module MAIN4, because you want to modify and recompile SOURCE.MAIN4 at a later date. You conserve public storage and increase the efficiency of the system by erasing all data sets not needed.

S,Y: Logoff

SYS:

The system confirms your LOGOFF command.

Example 5: Correcting and Recompiling a Prestored Source Program

In this example you modify and recompile the source program SOURCE.MAIN4 which, being on a public volume, was automatically cataloged for you in Example 4. You then run the new object module. After logging on, you issue:

S,Y: MODIFY SOURCE.MAIN4
SYS:

You enter a MODIFY command to alter the source program entered in the previous example.

S,Y: #
R,100

The system prints a number sign to request each modification. You wish to review line 100 before modifying it, and enter the R (for review), a comma, and then the line number.

SYS: 0000100 READ(5,10)A

The system prints the line. When you entered the statements, you separated them from their statement number with a tab character.

The system prompts for modifications with the number sign.

S,Y: #
100,1 READ(10,5,END=80)A

You modify the line by typing, following the number sign, the line number, a comma, and then the modified line. The system then prints another number sign.

S,Y: #
450, GO TO 1

You enter an entirely new line between 400 and 500.

S,Y: #
500,80 STOP

You add the label 80 as referenced in the READ statement.

S,Y: #
%E

You type these two characters to signal the end of modifications.

S,Y: CATALOG SOURCE.MAIN4, STATE=U, ACC=U, NEWNAME=SOURCE.MAIN5

You use the CATALOG command to rename the current source data set SOURCE.MAIN5. At compilation time, its list data set will be named LIST.MAIN5(0) by the system. The first U indicates the updating of an existing catalog entry. The second U specifies unlimited access for the data set.

S,Y: FTN MAIN5,Y,,Y

You enter the name and the rest of the FTN parameters you need: Y for prestored, the second comma after the first Y to default the version id parameter, and Y for ISD.

SYS: 0000100 F *** STATEMENT NUMBER USED AS FORMAT IS NOT DEFINED.

The compiler discovers an error in line 100. Note that this error was not detected during your use of the MODIFY command, since that command does no syntactical checking. You decide to correct the line.

SYS: 0000100 1 READ(10,5,END=80)A

The system prints the line that caused the diagnostic and asks whether you want to modify your source statements.

YOU: Y
S,Y: #100,1 READ(5,10,END=80)A
S,Y: # (press return key)

Having made the necessary correction, you signal that you are finished by pressing the return key. This is equivalent to the termination you indicated with %E during the MODIFY command. Then the compiler will rescan your source statements.

SYS:

The system informs you of the satisfactory completion of the compilation.

S,Y: PRINT LIST.MAIN5(0),PRTSP=EDIT,ERASE=Y

SYS:

The system acknowledges your PRINT command and informs you of the batch sequence number it has assigned to the printing task.

The compiled object module is stored in your USERLIB. You now proceed to run the modified program.

S,Y: CALL MAIN5

Since you gave no DDEF command for the data set reference number 5 in your READ statement, the system assumes you want to use the terminal for input.

SYS: (unlocks terminal keyboard)

YOU: 2.0

PGM: 2.00

6.28

Data set reference number 6 in your WRITE statement refers to the terminal for output.

SYS: (unlocks terminal keyboard)

YOU: 5.0

PGM: 5.00

15.70

SYS: (unlocks terminal keyboard)

YOU: 1.0

PGM: 1.00

3.14

SYS: (unlocks terminal keyboard)

YOU: %END

Satisfied that the program is now working correctly, you terminate execution by entering %END, which is the end-of-data indicator for SYSIN.

SYS: TERMINATED: STOP

Execution of MAIN5 being terminated, the system prints this message and then prompts for the next command.

S,Y: ERASE USERLIB(MAIN5)

You plan to recompile your program later, so you erase its object module from your USERLIB. This will prevent name duplication in the future, and conserve public storage space.

S,Y: LOGOFF

SYS:

The system confirms your LOGOFF command.

Example 6: Writing a Data Set and Printing It

In this example, you run a program that writes a data set that is much too long to be conveniently printed at your terminal. You run the program, and then request printing of the output data set on the high-speed printer as a separate task. After logging on, you issue:

```
YOU: (press attention or dial up system)
      LOGON ADUSERID,MYPASS*, ,ADACCT29
SYS: B001 LOGON TASKID=F206 12/15/69 10:12
S,Y: DDEF DDNAME=FT07F001,DSORG=VS,DSNAME=M6OUT,-
      DCB=(RECFM=FA,LRECL=80),DISP=NEW
```

In preparation for the execution of your program, you define a cataloged data set (M6OUT) for data set reference number 7. You specify a virtual sequential (VS) data set.

You specify that your record format is to be fixed-length (F) and the records contain print control characters (A). Its records are to be 80 bytes long.

```
S,Y: CALL MAIN6
```

The MAIN6 object module is stored in your USERLIB.

It ends with these statements:

```
WRITE(7,10)ALPHA,BETA,GAMMA
10  FORMAT(1H0,3F20.5)
      ENDFILE 7
      STOP
      END
```

The data set referred to in the WRITE7 and the ENDFILE7 statements is defined by the FT07F001 data definition.

```
SYS:  TERMINATED:  STOP
```

The system indicates the end of execution. The underscore on the next line indicates return to command made.

```
S,Y:  PRINT DSNAME=M6OUT,PRTS=EDIT
```

To print your newly written data set, this command creates a separate task similar to the tasks that have printed your listing data sets. Only the first operand, DSNAME=, must be specified; the remaining operands may be defaulted.

Explanation of PRINT Operands

STARTNO=

First Byte Position

You want printing to begin with the first byte of each data set record. You can enter a number consisting of one to six digits.

ENDNO=

Last Byte Position

This parameter specifies at which byte in each data set record printing is to end. Since your records are shorter than the default length of 132, your printing will end at the last (80th) byte of each record.

PRTS=

Spacing Option

Since you want line spacing to be controlled by the control character your program has supplied in each record, you choose EDIT. (The default would be 1.) Selecting EDIT requires that you default the next three parameters.

ERASE=

Erasure Option

This parameter is meaningful only if the data set being printed is cataloged. In that case, you can specify that the data set be erased after it is printed.

ERROROPT=

Error Option

This parameter applies only to data sets on tape. It specifies the action to be taken if an unrecoverable error is found while a data set record is being read.

FORM=

Type of Printer Paper

Here you can specify the kind of printer paper you desire for your output. The operands for this keyword are determined by your installation.

SYS:

The system informs you that it has created a separate task to print your data set.

S,Y: LOGOFF

SYS:

The system confirms your LOGOFF command.

Example 7: Reading and Writing Cataloged Data Sets

In this example, you run a program that reads one data set and produces two other data sets as output, as well as printing some short messages at the terminal. After logging on, you issue:

```
S,Y: DDEF FT11F001,VS,IN11
S,Y: DDEF FT22F001,VS,OUT22A
S,Y: DDEF FT22F002,VS,OUT22B
```

You give data definitions for the three data sets to be used.

A DDEF command remains in effect only for the session during which it was issued. Therefore, data definitions must be given even for data sets already cataloged. Data definition names for FORTRAN I/O have the standard form, FTxxFyyy, where xx is the data set reference number for an I/O statement, and yyy is the data set sequence number. You may use any data set name, as long as it is unique.

Because the first data set, IN11, was created previously, the system default for disposition is OLD. The next two data sets, OUT22A and OUT22B, are going to be created in this task, so they receive a system default disposition of NEW. Note that you do not have to include explicitly in the DDEF commands the parameter DISP=OLD for IN11, nor DISP=NEW for OUT22A and OUT22B.

The last two data definitions refer to the same data set reference number as in the FORTRAN WRITE statement, i.e., 22. Since an ENDFILE statement is given, and further WRITE statements are executed on the same data set reference number, two distinct data sets will be created. The second number in the data definition for the OUT22B data set name must be stepped from 1 to 2 to refer to the second data set written.

```
S,Y: CALL MAIN7
```

You run your FORTRAN object module, which was stored in your USERLIB in a previous session. MAIN7 includes these statements:

```
7      FORMAT (2F20.3)
10     READ  (11,7,END=80)A,B
      .
      .
      WRITE (22,7)A,B
      .
      .
      GO TO 10
80     ENDFILE 22
      .
      .
      WRITE(22,20)E,F
      .
      .
      ENDFILE 22
20     FORMAT(1X,2A4)
      WRITE (77,120)
90     STOP
120    FORMAT(26H FINISHED WRITING 2 FILES.)
      END
```

The last WRITE statement will produce a short message on the terminal.

```
PGM: FINISHED WRITING 2 FILES.
```

Because you did not issue a DDEF command for data set reference number 77, the system assumes you want your output from the WRITE statement at the terminal.

SYS: TERMINATED: STOP

The system indicates the end of execution with this message. The underscore mark indicates return to command mode.

S,Y: ERASE IN11

You erase the input data set that is no longer needed. Erasing a data set deletes its entry from your catalog and releases its storage space. You do, however, want to retain your two output data sets that were automatically cataloged for you at DDEF time.

SYS: LOGOFF

The system confirms your LOGOFF command.

Example 8: Multiple Compilation Before Execution

In this example you enter a BLOCK DATA program, then compile prestored main and subprograms. You create a new JOBLIB, which you catalog. After logging on, you issue:

S, Y: DDEF DDLIBA, VP, LIBA, OPTION=JOBLIB

With this data definition you create a new job library to hold the object modules from the three programs you are about to compile.

The system automatically catalogs LIBA as a new catalog entry and assigns an "unlimited" (read/write) access qualifier to the library.

S, Y: FTN BLKDATA8

You activate the FORTRAN compiler and specify the module name (BLKDATA8) for your BLOCK DATA program. You will enter your source statements at the terminal.

The system automatically catalogs your source data set as SOURCE.BLKDATA8.

```
S, Y: 0000100   BLOCK DATA
S, Y: 0000200   DIMENSION AB(3), AC(3)
S, Y: 0000300   COMMON/XY/AB/EXTRA/AC, ANSWER
S, Y: 0000400   LOGICAL ANSWER
S, Y: 0000500   DATA AB(1)/.007/, AB(2)/71.1/, AB(3)/8200.0/, AC/3*.88/, -
                .ANSWER/.TRUE./
```

The period preceding ANSWER is the continuation character and is therefore not part of the statement.

S, Y: 0000600 END

Your program initializes data for labeled common blocks XY and EXTRA.

SYS:

The system informs you of the satisfactory completion of the compilation.

S, Y: PRINT LIST.BLKDATA8(0), PRTSP=EDIT
SYS:

The system acknowledges your PRINT command and informs you of the batch sequence number it has assigned to the printing task.

S, Y: DEFAULT LPCXPRSS=Y

You indicate that you want the language processor "express mode" enabled. In express mode, several modules can be compiled in succession; the FTN command is issued for the first compilation, and only the module name is entered for succeeding compilations.

S, Y: FTN MAIN8, STORED=Y

SYS:

The system informs you of the satisfactory completion of the compilation and requests the name of the next module to be compiled.

S, Y: SUBR8

You enter the name of the next module to be compiled.

SYS:

The system informs you of the satisfactory completion of the compilation.

Y: _PRINT LIST.MAIN8(0), , , EDIT
SYS:

You enter a PRINT command preceded by a command system break character so that the system will interpret the line as a command. Express mode is turned off, and the system interprets the line as a command. The system acknowledges your PRINT command and informs you of the batch sequence number it has assigned to the printing task.

S,Y: PRINT LIST.SUBR8(0),PRTSP=EDIT
SYS: BSN=0569

The system acknowledges by issuing the batch sequence number of the PRINT job.

S,Y: LOAD BLKDATA8

A block data subprogram is the only program you are required to load. Main programs and subroutines they refer to are automatically loaded when you issue the RUN command.

SYS:

The system informs you that it has loaded BLKDATA8.
You must load the block data program prior to executing the main program.

S,Y: CALL MAIN8

Because you did not supply a DDEF for your WRITE statement in MAIN8, the output is received at your terminal.

PGM: 2048 VARIATIONS TRIED.
ANSWER IS F

SYS: TERMINATED: STOP

S,Y: ERASE SOURCE.BLKDATA8

You decide to erase your source data set. You retain your cataloged job library, LIBA, containing the three new object modules.

S,Y: LOGOFF

SYS:

The system confirms your LOGOFF command.

Example 9: Use of PCS Immediate Statements

In this example, you are executing a program for the first time. Since the program control system (PCS) provides complete debugging capability at execution time, you have not included any debugging aids in your compiled program. Anticipating the use of PCS, you requested an ISD when the source program was compiled.

S, Y: DDEF DEFJOB1, VP, JOB1, OPTION=JOBLIB, DISP=OLD

This command defines a job library JOB1, which contains the object module, MAIN9. JOB1 has been previously cataloged, but you must give this data definition to make it available. Note that you did not have to specify DISP=OLD explicitly because JOB1 was previously cataloged and the system defaults OLD in this case.

S, Y: CALL MAIN9

YOU: (press attention button)

You begin to run the program you wish to debug and then halt execution by pressing the attention button. The appearance of an exclamation mark indicates the system's readiness to accept new commands.

SYS:

YOU: QUALIFY MAIN9

After issuing this command, you can refer to internal symbols without the qualifying module name; they will be qualified automatically by the prefix "MAIN9."

S, Y: STOP

SYS: STOP AT MAIN9.86(4) PSW 2 0 0 004A3C12

The STOP command displays the FORTRAN statement number where the interrupt occurred, or, if the statement is not numbered, the most recent number plus an increment. In this case the (4) indicates that the interrupt occurred during execution of the third executable statement after statement number 86. The rightmost field of the PSW gives the virtual storage address of the next instruction to be executed. See Appendix B for more PCS information.

S, Y: LINE? SOURCE.MAIN9, (1200, 1400)

SYS: 0001200 7 ANGLEB=ANGLE1-ANGLE2

0001300 86 ALPHA=COS(ANGLEA)

0001400 BETA=SIN(ANGLEB)

You request a printout of source lines 1200 to 1400 which you believe include statement number 86. (Note the distinction between line numbers, which are not part of the program, and statement numbers, which are.) To obtain a printout of the current values of the variables ALPHA and BETA, you issue a DISPLAY command.

S, Y: DISPLAY ALPHA, BETA

SYS: ALPHA=+.17751984E+00

BETA=+.00000000E+00

S, Y: SET BETA=1.0

Since you previously issued a QUALIFY command for MAIN9, you specify only the internal names of the variables. Using this new information, you decide to change the value of a key variable to determine if the program will run to successful completion. You change the value of BETA and the system informs you of the new value.

S, Y: GO

You next enter a GO command. It causes execution to resume from the point of interruption.

SYS: TERMINATED: STOP

The program runs to completion.

S,Y: LOGOFF
SYS:

The system confirms your LOGOFF command.
Alterations you made with PCS commands exist in your program only while it is executing in virtual storage.

Since PCS alterations do not affect your object module, permanent changes should be made by modifying the source statements and then recompiling.

Example 10: Use of PCS Dynamic Statements

In this example, you use some of the more powerful commands of pcs. pcs provides trace facilities, conditional program interruptions and modification of variables, and dumps. After logging on, you issue:

S,Y: DDEF DDCURR,VP,CURRENT,OPTION=JOBLIB

This DDEF command causes your job library CURRENT to be placed at the top of your program library list. CURRENT has been previously cataloged and contains compiled object modules.

S,Y: ERASE PCSOUT10

S,Y: DDEF PCSOUT,VI,PCSOUT10

The DDEF command defines the data set that will be filled by the PCS DUMP command; the data set can later be printed. It requires the data definition name PCSOUT and virtual index sequential (VI) organization. You name the data set PCSOUT10. You precede the DDEF command with an ERASE command to ensure that PCSOUT10 does not contain any data before the DUMP command is issued.

S,Y: LINE? SOURCE.MAIN10,2100,2700,3200
SYS: 0002100 A=ATAN(Y)
0002700 101 VAR=A**2
0003200 210 GO TO(301,302,303,304),J

You decide to display three of your source statements in MAIN10, which has been previously compiled and cataloged. With the compilation, you had requested an internal symbol dictionary (ISD).

S,Y: LOAD MAIN10
SYS:

The system informs you that it has loaded MAIN10. In addition to its use for loading block data subprograms, you must use the LOAD command if you wish to enter a PCS statement before execution begins. Since the LOAD command does not initiate execution, you must eventually issue a GO or RUN command.

S,Y: QUALIFY MAIN10

The QUALIFY command enables you to designate, before referring to a group of internal names, the module in which these names are defined; thereafter, you may refer to these names without explicitly qualifying them by module name.

S,Y: AT 101; IF A>B & J=1; SET X=1.0
SYS: 00001

This AT command will cause a message to be printed on SYSOUT when execution of MAIN10 reaches statement number 101 and the IF condition is true. In addition, the IF and SET commands will cause the following: if A is greater than B at that time, and J is equal to 1, then X will be set to 1. Execution will then proceed.

The system assigns a number to each command containing an AT statement (here 1) that can be used later for removing the statement.

S,Y: AT 210 STOP
SYS: 00002

You also request that execution be stopped when it reaches statement number 210. Without the QUALIFY command you would have had to write MAIN10.101 in the first AT and MAIN10.210 in this one.

S,Y: MAIN10
SYS: AT MAIN10.101 PSW 3 0 0 005F2AB0 1
STOP AT MAIN10.210 PSW 4 0 0 0067D238 2

You execute the program. Your IF condition is fulfilled, X is set equal to 1, and your program stops at statement 210. If the IF condition were not satisfied, the SET would not be performed and you would not receive the X= printout. The number 1 appearing at the end of the PSW output is the PCS statement number assigned by the system.

S,Y: DUMP MAIN10#C, MAIN10#P
 You request a formatted dump of MAIN10's CSECT and PSECT. It will be written in the PCSOUT10 data set that you defined earlier.

S,Y: RELEASE PCSOUT
 If you wish to print the data set during this session, you must first issue a RELEASE command for its data definition. This causes the data set to be closed.

S,Y: PRINT PCSOUT10,ERASE=Y
 The data set will be erased after it is printed.

SYS:
 The system acknowledges your PRINT command.

S,Y: REMOVE 1,2
 This command deletes the previously issued PCS statements that include AT commands.

S,Y: GO
 You now resume execution of your program.

SYS: TERMINATED: STOP

S,Y: LOGOFF
 The program runs to completion, and you logoff.

SYS:
 The system confirms your LOGOFF command.
 Alterations made to your program with the PCS commands (SET, AT) exist only in virtual storage. To make **permanent** changes to a program, reassemble from an altered source data set. This causes the changes to be incorporated into the object module, which you would then load.
 Changes you make with the SET command remain in effect as long as the program is loaded. (By contrast, all AT commands in **any** of your programs are completely removed if you implicitly or explicitly unload a module that is referred to by any AT command.)
 Logging off causes all of your programs to be unloaded from virtual storage.

Example 11: Input and Output on Tape

In the previous examples, all of your data sets resided on direct-access devices (disks) which were assigned to public storage. In this example, your data sets reside on tapes, which are always private volumes.

You will run a previously-compiled program that reads a data set from a labeled tape and processes the input data. Then it writes a new data set on another tape. After logging on, you issue:

S,Y: DDEF FT01FO01,PS,SAMPLE01

This command defines the data set that your program is to read. Since it is cataloged, you need enter only these parameters. Omitted information about the data set's characteristics (record format, record length, organization) will be obtained from the tape label that was created by the system when the data set was written. Information about the volume on which the data set resides (9-track tape, private, volume serial number, etc.) will be provided from the catalog entry.

SYS:

The system will inform you that the task is waiting for volume mounting. You will be informed by the system when the wait is over.

S,Y: DDEF FT02FO01,PS,RESULT01,-
UNIT=(TA,9),LABEL=(,SL),VOLUME=(PRIVATE)

Here you define your output data set. It is not yet written, and is not cataloged, so you must supply all the necessary DDEF parameters.

Because it is not yet written, the disposition field is defaulted to NEW. The data set is to have physical sequential (PS) organization, is to reside on a 9-track tape, and is to be provided standard labels (SL) by the system.

By omitting the DCB field, you select the default options of variable length and unblocked records. You do not specify the volume serial number in the VOLUME field, so the system instructs the operator to choose a tape reel from the installation pool. (Refer to Appendix E for further details on specifying DDEF parameters.)

SYS:

The system will inform you that the task is waiting for volume mounting. Again you must wait until the tape is mounted.

S,Y: CALL MAIN11

You execute your object program, which was stored in your USERLIB.
MAIN11 has the following significant I/O and related statements.

```
      .  
      .  
      DIMENSION SAMPLE(250),RESULT(250)  
10    FORMAT (250A4)  
      .  
      .  
100  READ (1,10,END=900, ERR=800)SAMPLE  
      .  
      .  
      CALL TRNFRM (SAMPLE,RESULT,250)  
      .  
      .
```

```
        WRITE (2,10) RESULT
        .
        .
        .
        GO TO 100
800    .
        .
        .
        GO TO 100
900    ENDFILE 2
        STOP
        .
        .
        .
        END
```

SYS: TERMINATED: STOP
Your program concludes.

S,Y: CATALOG RESULT01, STATE=N, ACC=R

This command causes your output data set to be cataloged, thus recording its characteristics and volume serial number in the catalog. You will still have to issue a DDEF command in order to use this data set in a later session, but the system will retrieve its characteristics from the catalog, so at a later session, a DDEF of the following form will suffice:

DDEF ddname,,DSNAME=RESULT01

The N indicates that the catalog entry is new (data set not currently cataloged).

You want to protect this new data set from accidental destruction in a later session so you restrict the access to read only (R).

S,Y: LOGOFF
SYS:

The system confirms your log-off request.

The private volumes (your two tapes) are demounted by the operator, and retained at the installation.

This example can be run only in conversational mode. To run it noncon conversationally, you would omit from the LOGON command your password, which is only used conversationally, and add another command: SECURE (TA=2,9).

SECURE must appear immediately after the LOGON command. It would inform the system of device requirements (here, two 9-track tape units) prior to execution of the nonconversational task. Such tasks are described in later examples.

Example 12: Conversational Initiation of Nonconversational Tasks

It is often more convenient to have your programs run after you have left the terminal, that is, to have them run in nonconversational mode. Two ways of doing this after logging on are shown in this example.

In Part 1, you begin your task conversationally and then use the `BACK` command to switch its execution to the nonconversational mode.

In Part 2, you construct a nonconversational task and then use the `EXECUTE` command to cause it to be executed at a later time.

Part 1: The `BACK` Command

```
S,Y: DATA PROC12A
```

With this command you build the `SYSIN` data set (named `PROC12A`) that will provide input to your task after you have switched to the nonconversational mode. You do not need to issue a `DDEF` command for the data set created by a `DATA` command. By default, the data set organization is `VS`.

```
S,Y: #DDEF FT09F001,VS,SPRING
S,Y: #CALL MAIN12
```

The system prompts (with `#`) for the first command to be executed in your nonconversational task. This `DDEF` command defines the new data set for data set reference number 9. It is to reside on public storage and is therefore automatically cataloged for you.

`MAIN12` contains a “`READ(1,m)` list” and a “`WRITE(9,n)` list” statement. You omit a data definition for data set reference number 1 because you will provide input data in `PROC12A`, which will be used as `SYSIN`.

```
S,Y: #(enter data to be read by MAIN12)
      .
      .
      .
```

The `DATA` command accepts each line as a string of characters. Any mistakes you make while creating this data set will not be detected until the `BACK` command is executed.

```
S,Y: #%END
```

When `%END` is read from `SYSIN`, it indicates the end of data to your program.

```
S,Y: #LOGOFF
S,Y: #%E
```

With the `%E` you indicate that your data set is complete. Now you are prompted with an underscore.

```
S,Y: BACK PROC12A
SYS:
```

The system informs you that your `BACK` command has been accepted. Your `BACK` command has been accepted and the task will be continued immediately as a nonconversational task beginning with the `DDEF` command. (Note that `DDEF` commands for **private** volumes must be given prior to issuing the `BACK` command.) Should you wish to cancel the task you would issue a `CANCEL` command which specified the batch sequence number.

Now you can depart and let the task run, since `PROC12A` is now its `SYSIN` and includes a `LOGOFF` command for task termination. If you wish to initiate another task, you must log on again.

The `BACK` command may not complete its operation if the attention key is depressed shortly after issuing the command. The result is a nonconversational task still connected to a terminal. Wait a few seconds before initiating logon procedures.

Part 2: The EXECUTE Command

```
S,Y: DATA PROC12B
S,Y: #LOGON ADUSERID,,ADACC29
```

The LOGON command is the only difference between this and the PROC12A SYSIN data set created in Part 1. Since the task whose commands and data are stored in PROC12B will be run later instead of being continued, you must provide a LOGON command.

Note that a nonconversational LOGON omits the password. The remainder of PROC12B is the same as in Part 1.

```
S,Y: DDEF FT09F001,VS,SPRING
S,Y: #CALL MAIN12 (same program as in Part 1)
S,Y: # (enter data to be read by MAIN12)
```

```
·
·
·
```

```
S,Y: #%END
S,Y: #LOGOFF
S,Y: #%E
S,Y: EXECUTE PROC12B
SYS:
```

The system informs you that your EXECUTE command has been accepted. Your request for a nonconversational task has been accepted by the system, and will be executed when system resources are available. The SYSOUT of this task will consist of system messages and any output to SYSOUT generated by your executing programs.

Because the terminal is active (you are still logged on) after an EXECUTE is issued, another command sequence can be entered. In fact, another sequence similar to the one illustrated could be issued to create other tasks.

```
S,Y: LOGOFF
SYS:
```

The system confirms your LOGOFF command.

Example 13: Preparing a Job for Nonconversational Processing

In this example, you put a series of commands and input data on cards. You will subsequently send them directly to the installation operator, who will store the information from the cards into a data set. The data set will then become the SYSIN for a nonconversational task (described in the cards) and will be queued for execution.

CARDS

```
LOGON ADUSERID,,,ADACCT29
```

When entered on a card, the LOGON command must start in the **third** card column, and the first two columns must be blank. All the required LOGON parameters must be included in the same card. The password is not used.

```
DDEF DDNAME=SCRATCH,VP,SCRATCH,OPTION=JOBLIB
```

The command will define a new JOBLIB on which to store the object module to be created by the compiler. SCRATCH will automatically be cataloged for you by the system.

```
FTN MAIN13,LISTDS=Y
```

Here you use the LISTDS operand, which works only in nonconversational tasks. (If used in a conversational task, it is ignored.) By specifying LISTDS=Y, you cause the listing to be placed in the list data set, as in conversational tasks. If you did not specify LISTDS=Y, the listing would be printed automatically; then it would no longer exist in the system.

Your source statements follow. Note that the requirements for direct input of a FORTRAN source program at the terminal keyboard and on cards are not the same; here you must conform to FORTRAN source coding format. This is discussed more fully in Appendix A.

Your source program, SOURCE.MAIN13, will automatically be cataloged for you by the system.

```
10 READ(5,10)A
   FORMAT (F6.2)
   ATC=A*3.141
   WRITE(6,10)A,ATC
   STOP
   END
```

After compilation, the object module will reside on the library at the top of your program library list, in this case, the job library SCRATCH. You do not issue a PRINT command in this task. However, the listing data set procedure is retained as the latest generation of LIST.MAIN13, and you can later print it if you wish by issuing the following command: PRINT LIST.MAIN13(0),, EDIT.

```
CALL MAIN13
```

This command will initiate execution of your newly compiled module.

```
099.70
```

This card contains your object time data.

```
%END
```

The %END will signal the end of the program data. The % character must be in column 1 of the card. This card must immediately follow the last input data card. This card activates the "END=" option on a READ statement, if you have used it in your program; if you have not, it initiates proper termination of the program execution when a READ is executed after the end of data has been reached. If your program depends on either of the above, and you have omitted the %END card, your program will read the subsequent commands as SYSIN data, producing abnormal results.

```
LOGOFF
```

Enter LOGOFF beginning in column 3.

Three things in particular should be kept in mind when preparing a deck of cards for processing:

1. Although the positioning of characters when typed in directly can be "free form," the positioning on cards is more closely fixed.
2. Any errors in preparing the deck will probably terminate the task, since the system cannot prompt you for corrections.
3. The "modifications" and "continue" compilation prompts will not occur, so no Y or N responses should be specified.

Example 14: Storing DDEF Commands for Later Use

In Part 1 of this example you create a data set containing DDEF commands for frequently used data sets. In Part 2 you cause them to be issued with a CDD (call data definition) command. After logging on, you issue:

Part 1: Storing DDEF Commands

S,Y: DATA DDPACK.MAIN14,RTYPE=I,BASE=1000,INCR=400

The DATA command can be used to store any kind of information that can be transmitted through the terminal. Here you are going to use it to store your DDEF commands in a data set you name DDPACK.MAIN14. The commands are stored as character strings but are interpreted as commands when they are later retrieved with the CDD command.

The I specifies that the data set is to be indexed. The first line number is to be 1000, and succeeding line numbers are to be incremented by 400. Default values for each of these parameters is 100.

The data set, DDPACK.MAIN14, is automatically cataloged for you by the system.

S,Y: 0001000 DDEF YOURLIB,VP,MAINPGMS,OPTION=JOB LIB

The system prompts you for each line with a line number. You enter the first DDEF to be stored. It is for the cataloged job library that contains your compiled program to be run, MAIN14.

The DDEFs do not have to be stored in any special order in the data set, but their ddnames must be unique.

S,Y: 0001400 DDEF FT01F001,VS,IN14

Data set reference number 1, input for MAIN14, has been cataloged in an earlier session under the name IN14.

S,Y: 0001800 DDEF FT09F001,PS,DATA,UNIT=(TA,9),LABEL=,-
S,Y: 0002200 (,SL),VOLUME=(,012170)

MAIN14 also expects input on data set reference number 9 from an uncataloged data set residing on tape 012170. Note the use of hyphen to continue the command on the next line.

S,Y: 0002600 DDEF FT51F001,VS,OUTPUT.MAIN14

Since MAIN14 output will be too large to be conveniently printed at the terminal, this cataloged data set is to be defined to hold it.

S,Y: 0003000 %E

The %E indicates that input to the DATA command is complete.

S,Y: LOGOFF
SYS:

The system confirms your LOGOFF command. Your program, MAIN14, and its associated DDEF commands are ready for use. You now check them out, to be sure there are no errors.

Part 2: Retrieving Stored DDEF Commands

After logging on, you issue:

S,Y: CDD DDPACK.MAIN14,YOURLIB

You cause the DDEF that defines your job library to be issued.

SYS: 0000, DDEF YOURLIB,VP,MAINPGMS,OPTION=JOBLIB

The system executes the specified DDEF command, prefixing four zeros to distinguish it on your SYSOUT listing from those DDEF commands entered directly through SYSIN. Any diagnostic would be printed at this point, as the system is now analyzing the character string as a DDEF command.

S,Y: CDD DDPACK.MAIN14,(FT01F001,FT09F001,FT51F001)

This command causes it and all the remaining data definitions to be executed. You may enter any number of ddnames. Omitting this parameter causes all the DDEF commands to be executed.

SYS: 0000 DDEF FT01F001,VS,IN14
0000 DDEF FT09F001,PS,DATA
UNIT=(TA,9),LABEL=(,SL),VOLUME=(,012170)

SYS:
The system informs you that it is waiting for volume mounting.
You must wait until the operator mounts your tape.

SYS:
The system informs you that the task is now proceeding and the wait is over.

SYS: 0000 DDEF FT51F001,VS,OUTPUT.MAIN14
You begin execution of a program.

S,Y: CALL MAIN14

SYS: TERMINATED: STOP
Execution of your program is complete. The output of MAIN14 went to the data set OUTPUT.MAIN14.

S,Y: PRINT OUTPUT.MAIN14,PRTSP=EDIT

SYS: The system acknowledges your PRINT command.

S,Y: LOGOFF

SYS:
The system confirms your LOGOFF command.
The following card deck is all you need to run this task nonconversationally.

CARDS

```
LOGON ADUSERID,,ADACCT29
SECURE (TA=1,9)
CDD DSNAME=DDPACK.MAIN14
RUN LOC=MAIN14
PRINT OUTPUT.MAIN14,PRTSP=EDIT
LOGOFF
```

Example 15: References to Subroutines

In this example you run two programs, MAIN15 and MAINXV, each having references to other programs. After MAIN15 is loaded, a diagnostic warns you of an undefined reference. You ignore it and execute anyway. After MAINXV is loaded, a diagnostic warns you of an undefined reference in a program called by MAINXV. In this case you resolve the reference before executing. After logging on, you issue:

S,Y: CALL MAIN15

The CALL command causes the system to load the specified object module and to initiate its execution. During loading, the loader attempts to resolve any external references by searching the libraries in the program library list and loading those modules with definitions that satisfy the references. These new modules may in turn have references to other modules.

SYS: *****UNDEFINED REF (ACLRTN) IN MODULE (MAIN15).ADDRESS FFFFF000 ASSIGNED
F025 MAIN15 ERROR IN LOADING MODULE
F002 STATEMENT REJECTED

Here a diagnostic is issued because the loader could not resolve a CALL for ACLRTN. It assigns an invalid address that will cause an interrupt if it is executed. Since you are certain that in this case the actual execution flow of MAIN15 will not go through the CALL on ACLRTN, you decide to go ahead and execute.

S,Y: GO

You issue a GO command, but the system merely prints an underscore.

S,Y: CALL MAIN15

You then issue the CALL command with the required module name. This must be done after the occurrence of the F002 diagnostic.

SYS: TERMINATED: STOP

Your program runs to completion. ACLRTN was not called during execution. If it had been, an invalid address diagnostic would have appeared and an abnormal task termination would have followed.

Now you want to run MAINXV. You have to decide whether or not to unload MAIN15. If you don't unload it, any programs already loaded for MAIN15 that MAINXV also uses will not have to be reloaded, thereby saving some time. On the other hand, if the programs are not unloaded and then reloaded, the loader will not be able to check their references, and any unresolved references they might have will go undetected. To be safe, you unload MAIN15.

S,Y: UNLOAD MAIN15

MAIN15 and any other modules that were loaded because of references from it are unloaded.

S,Y: CALL MAINXV

Now you initiate the loading and execution of MAINXV.

SYS: *****UNDEFINED REF (PRESSR) IN MODULE (THRUST44).ADDRESS FFFF0000 ASSIGNED
F025 THRUST44 ERROR IN LOADING MODULE.
F002 STATEMENT REJECTED.

THRUST44 was loaded because of a CALL to it from MAINXV. THRUST44 in turn contained a CALL to PRESSR, which could not be located after a search of all the libraries on the program library list. You remember that a job library named ROCKETS has an object module which contains a definition for PRESSR.

S,Y: DDEF MYLIBE,VP,ROCKETS,OPTION=JOB LIB,DISP=OLD

You issue a DDEF for the library ROCKETS to put it at the top of the program library list.

S,Y: UNLOAD MAINXV

Then you unload MAINXV from virtual storage.

S,Y: CALL MAINXV

Reloading MAINXV causes the loader to search the libraries in the program library list again and it resolves the CALL to PRESSR from the newly defined library and loads the module containing the definition of PRESSR.

SYS: TERMINATED: STOP

S,Y: LOGOFF

After execution is complete, you log off.

SYS:

The system confirms your LOGOFF command.

Example 16: Entering Data for Later Use

In this example, you create two data sets, one containing a source program for compilation in a later session, the other containing data to be read by the program stored in the first data set. After logging on, you issue:

S,Y: DATA SOURCE. ORBIT, I, 2000, 200

SOURCE. ORBIT is the name of the data set you are about to create. When the DATA command is used, the system automatically supplies a data definition. The qualifier "SOURCE" is needed because this data set is intended to be used as input to the FORTRAN compiler. In earlier examples, where source statements were entered after a FTN command, the source data set was created automatically and the qualifier, SOURCE, was attached by the system.

The I specifies that the data set is to be indexed. The first line number is to be 2000, and succeeding line numbers are to be incremented by 200. Default values for each of these parameters is 100.

The source data set, SOURCE. ORBIT, is automatically cataloged for you by the system.

S,Y: 0002000C ORBIT CALCULATIONS, HIGH ECCENTRICITY

The DATA command prompts for input by printing a line number. You enter a comment line (C in column 1). You may start typing immediately after the line number, or use a tab stop to format the terminal printing of your input. The tab stop may be set at this time. When setting your tab stop, you should issue a line cancellation (a pound sign at the end of the line) to cancel any of the spurious characters (tab, space, etc.) that were generated when you set your tab.

S,Y: 0002200 DIMENSION A(100,100), B(1000)

S,Y: 0002400 REED (5,10)A

You notice you made a mistake in line 2400.

Since the line has not yet been completed (you have not pressed the return key) you cancel the line by using the # sign. You then tab over and enter the correct line.

YOU: READ (5,10)A

S,Y: 000260020 FORMAT (19H PERTUBAT

RBATION TABLE)

You make another error, but this time you correct only the part in error. You back-space 3 times, cancelling BAT, move the paper up once manually to avoid overtyping, and continue the line correctly.

S,Y: 0002800 CALL THRUST(A, B)

S,Y: 0003000%2200, DIMENSION A(100,100), B(1000)

You notice another misspelling made earlier and decide to correct it by replacement. The percent sign in the first typeable position has a special meaning to DATA when of the form: %line number. It means that an insertion or replacement is to be made. If the line number specified already exists, a replacement by the character string following the comma will be made. If the line number falls numerically between two existing ones, an insertion will be made at that point.

In this correction and in the one after the next, the tab stop cannot be used because of the double line numbers. Therefore, to maintain uniformity for future printouts of your source data set, you enter six spaces between the comma and the source statement.

S,Y: 0003000 WRITE (6,10)B

The correction is made and you are again prompted for line 3000.

S,Y: 0003200%2700,30 FORMAT (E13.6)

Instead of entering a line for 3200, you insert a new line between 2600 and 2800.

S,Y: 0003200

·
·
·

S,Y: 000700010 FORMAT (1X,2F10.6)

When entering your VS input data for your program via the DATA command (see below), the system places a keyboard-card character for internal use prior to each FORTRAN record. In this example, then, your FORMAT statement must include an X specification to skip one character for each record to be read so that your program will skip over the unwanted system character.

S,Y: 0007200%D,2600

The %D deletes line number 2600 from the data set. If a range of lines is to be deleted, two line numbers are specified; the first line number must be lower than the second.

S,Y: 0007200 END

S,Y: 0007400%E

The %E indicates to the DATA command that your data set is complete.

S,Y: DATA TELEMO01

You enter another DATA command, this time defaulting to no index, which means you will create a virtual sequential (VS) data set. This data is to be read by the program you just entered. Because this data set also resides on public storage, the system automatically catalogs TELEMO01 for you at this time.

S,Y: #099.900000-08.732460

S,Y: #093.247650-01.178940

·
·
·

When a sequential data set is specified, line numbers do not appear; instead the system prompts you for each line with the number sign. The tab key has not been used in entering this data because it would then not be compatible with the input FORMAT statement in ORBIT. Since a special character code is transmitted each time the tab is depressed, allowance for use of the tab key would have to be made when writing the FORMAT statement by using the X specification to skip one character for each time you planned to press the tab key.

S,Y: #%E

S,Y: RET TELEMO01,PR

When a data set with virtual organization residing in public storage is automatically cataloged for you, the system creates a new catalog entry and assigns an access qualifier of "unlimited" (read/write) to the data set.

You wish to change the system assigned access qualifier because this data set contains important information which you do not want accidentally destroyed. You therefore issue a RET command giving the data set name TELEMO01. The operand after the data set name specifies the data set is to reside on permanent storage and have the read-only attribute.

S,Y: LINE? SOURCE.ORBIT, (2000,7400)

To check on the corrections made while entering your source statements, you use this command to print the contents of the data set you have just created. You specify the first and the last lines to cause the printing of the entire data set.


```

SYS: 0002000 C    ORBIT CALCULATIONS, HIGH ECCENTRICITY
      0002200    DIMENSION A(100,100),B(1000)
      0002400    READ (5,10)A
      0002700 30  FORMAT(E13.6)
      0002800    CALL THRUST(A,B)
      0003000    WRITE(6,10)B
      0003200
      .
      .
      .
      0007000 10  FORMAT(2F10.6)
      0007200    END
D334 LAST LINE IN MEMBER OR DS IS 7200
      0007200    END

```

SYS:

The system informs you that the last line of the data set is line numbered 0007200. The space between the line numbers and the source statements is created by the system. It indicates that the lines were originally entered from a terminal keyboard. Lines entered from the terminal card reader are indicated by a C in that space.

S,Y: LOGOFF

SYS:

The system confirms your LOGOFF command.

Tab stops were used to make the terminal printout format resemble FORTRAN source card format. Note that, just because your terminal printout format looks like card images, it does not mean that cards punched from this data set can be used as FORTRAN source input. In this example they cannot. For a further discussion of punching source cards from a data set prepared at the terminal, see Appendix A.

Example 17: Data Set Considerations When Interrupting a FORTRAN Execution

In this example, you run a program that uses a data set on a private disk as input. When execution begins, you realize you have specified the wrong data set. To start over you release the data set and unload your program. You then run with the correct data set. After logging on, you issue:

S,Y: DDEF FT11F001,VS,RESULT01,OLD

This command defines the data set that provides input to your MAIN17 program. DCB subparameters are not required for existing data sets on direct-access devices, since the data set characteristics were recorded with the data set itself when the data set was created.

SYS:

The system informs you that the task is waiting for volume mounting. You must wait for the operator to mount your disk.

S,Y: DDEF FT05F001,VS,TABLE001

This command defines a new data set on public storage that is to contain some of your program output. It is automatically cataloged for you by the system.

S,Y: CALL MAIN17

•
•
•

You default data set reference number 1, which was referred to in MAIN17, to SYSOUT (terminal).

PGM: FIRST RECORD OF TEST RESULTS SHOWS
 DATE OF TEST — 11/17/69
 LOCATION A
 ALTITUDE RANGE 20000.0 TO 30000.0

This is some of the output from MAIN17.

YOU: (press attention button)

Realizing that you have specified the wrong data set and volume number in your DDEF command, you stop execution of MAIN17 by pressing the attention button.

SYS:

S,Y: UNLOAD MAIN17

You make sure that all your data sets are closed. Remember that the data set TABLE001 was to be created during this run. If some output had gone to this data set before you pressed the attention button, and another RUN MAIN17 was issued without unloading, whatever had been written in TABLE001 would be left there after restart.

S,Y: RELASE FT11F001

This command releases the DDEF command previously issued. You must issue it before the FT11F001 data definition can be repeated for another data set.

S,Y: DDEF FT11F001,VS,RESULT02,OLD

You define what you believe to be the correct data set.

SYS:

The system informs you that the task is waiting for volume mounting. After being notified that the operator has mounted disk number 012237, you again execute your program.

S,Y: CALL MAIN17

•
•
•

You default data set reference number 1, which was referred to in MAIN17, to SYSOUT (terminal).

PGM:

FIRST RECORD OF TEST RESULTS SHOWS
DATE OF TEST — 11/23/69
LOCATION B
ALTITUDE RANGE 5012.9 TO 6492.3

Everything now appears to be all right, so you allow execution to proceed to the exit message and underscore.

SYS: TERMINATED: STOP

S,Y: LOGOFF

The system confirms your LOGOFF command.

Example 18: Sharing Data Sets

This example shows how data sets can be shared by several users of the system. Part 1 shows a session during which another user makes one of his data sets available to you. Part 2 shows how you copy the data set so that you can make changes to it.

Part 1: Permitting Access to a Data Set

After logging on, user ABPALID issues:

S,U: PERMIT DATA,USERID=ADUSERID,ACCESS=RO

He makes available to you his cataloged data set DATA with read-only access point.

S,U: RET DATA,R

He changes his own access to DATA to read-only.

S,U: DDEF FT01FO01,VS,DATA

The user defines data set DATA for his new task; he now has read-only access to it.

S,U: PROG1

He executes PROG1, a program that uses data set DATA.

USR: LOGOFF
SYS:

The system confirms his LOGOFF command.

Part 2: Accessing a Shared Data Set

After logging on, you issue:

S,Y: SHARE DATA,USERID=ABPALID,OWNERDS=DATA

An entry will be created in your catalog for data set DATA. This command would have been rejected if the owner had not previously granted you access with the PERMIT command.

S,Y: DDEF FT03FT001,VS,DATA

You issue a DDEF command defining DATA for your task.

S,Y: PROG9

You can now execute PROG9, a program that uses data set DATA.

Note: FORTRAN I/O opens all virtual storage data sets for OUTPUT unless the data set is read-only; read-only data sets are opened for INPUT. If a shared data set is opened for OUTPUT, an interlock is set that prevents other users from having access to the data set until it is closed.

YOU: LOGOFF

You must remember that if the owner erases or deletes one of his data sets which you share, its entry in your catalog is not removed. To remove the entry from your catalog, you must issue a DELETE command.

SYS:

The system confirms your LOGOFF command.

Example 19: Manipulation of Several Forms of a Program

In this example, you examine a previously cataloged program. Then you remove all forms of it from the system. After logging on, you issue:

S,Y: LINE? SOURCE.MAIN19,(1,5000)

You would like to eliminate from the system any forms of a program named MAIN19 that you no longer need. You want to punch the source data set on cards, but first you must determine whether such cards can be used as compiler input. So you issue this command in order to examine the source data set.

SYS:

The system informs you that the first line number in the data set is 000100. The system then proceeds with the actual listing.

SYS:

```
0000100CC THRUST CALCULATIONS FOR MK.1 ENGINE
0000200CC WITH STANDARD ATMOSPHERE.
0000300C      DIMENSION ATBL(10,1000),FORM(10)
0000400C10     FORMAT (10F10.6)
0000500C20     FORMAT (10A4,2I4)
0000600C      READ (3,20)FORM,I,J
0000700C      READ (3,FORM) (ATBL((II,JJ),II=1,I),JJ=1,J)
0000800C      WRITE (4,10)ATBL(I,
```

YOU: (press attention button)

Satisfied that the program is what you want, you terminate the LINE? command. The C following the line numbers indicates the statements were originally entered via a card reader. This means that, if you punch the source data set, the cards can later be used for compiler input. (See Appendix A for a more detailed description of compiler input format requirements.)

S,Y: LINE? SOURCE.MAIN19,9999999

SYS:

The system issues the last line in the data set as: 0013700C END

In order to determine the size of the data set, you request the maximum possible line number. This causes the last line of a line data set to be printed.

S,Y: PRINT SOURCE.MAIN19

This command causes the creation of a nonconversational task that will print the source data, and issues a batch sequence number for the nonconversational task.

SYS: PRINT BSN=0375

The system acknowledges your PRINT command and assigns it a batch sequence number (BSN).

S,Y: PUNCH SOURCE.MAIN19,STARTNO=9,ENDNO=88,ERASE=Y

You request the punching of the 9th through 88th characters of each record in your source data set. The first eight characters are the line number and the input key. The system creates a separate task that will perform the punching when system resources are available.

You had to use the ERASE option in the PUNCH command, rather than a separate ERASE command following it. The system will reject an ERASE command if the data set referred to has an associated print or punch task pending. Most likely the two tasks will be executed in the same order as they were entered. It is possible, however, that they actually will be executed in reverse order. If so, the ERASE option will be delayed until after the PRINT task has been completed. You should not insert the ERASE option until the last print or punch data command in any sequence which refers to the same data set. It is possible that the first one, for example the PRINT above, could be executed in less time than it takes to type in the next command; therefore the ERASE option on the PRINT command could take effect before the PUNCH task could be executed.

S,Y: CANCEL 0375

You now decide that a listing of the source data set is superfluous (since you will have a source deck), so you cancel the printing task, referring to it by the batch sequence number.

S,Y: ERASE LIST.MAIN19(-1)

Here you erase the generation that contains the earlier version of MAIN19.

S,Y: ERASE LIST.MAIN19(0)

Here you remove the latest generation of LIST.MAIN19 which you created and cataloged when the program was compiled in an earlier session.

S,Y: ERASE USERLIB(MAIN19)

This command erases the last form of this program, the object module produced during compilation.

S,Y: LOGOFF

SYS:

The system confirms your LOGOFF command.

Example 20: Terminal Input of a Pre-Punched Program for Compilation and Running

In this example, you use a terminal that has a card reader. You switch back and forth between terminal keyboard input and card input as you compile and execute a program.

You place a deck in your terminal card reader. LOGON and LOGOFF command cards must begin in column 3, and their first two columns must be blank. All other commands may begin in column 1.

CARDS

```
FTN MAIN20,ISD=Y,LISTDS=Y
C   MATRIX I/O AND MULTIPLY
   DIMENSION A(20),B(20),C(20),FORM(10)
10  FORMAT (10A4,I3)
15  READ(7,10)FORM,I
20  READ(7,FORM,END=30)(A(J),J=1,I),(B(K),K=1,I)   Source
   CALL MATMPY(A,B,C,I,I,I)                       Statements
   WRITE(17,FORM)(C(J),J=1,I)
   GO TO 20
30  STOP
   END
PRINT LIST.MAIN20(0),PRTSP=EDIT
KB  Return to keyboard
```

After logging on, you issue:

S,Y: CB (press reader start button)

This causes SYSIN to be switched to the card reader for one card. You can do this any time the system is waiting for keyboard input if the desired cards are ready in the card reader.

Since the terminal is in send-receive mode, each card image is printed on the terminal as if it had been typed in at the keyboard.

CIP: LOGON ADUSERID,MYPASS*,.,ADACCT29

This is the card image printout (CIP). You supply the LOGON command from the card deck.

S,Y: CB (press reader start button)

S,C: FTN MAIN20,ISD=Y,LISTDS=Y

The system prompts for source statements by issuing a line number. It then reads the statements from the card reader.

```
S,C: 0000100C   MATRIX I/O AND MULTIPLY
S,C: 0000200   DIMENSION A(20),B(20),C(20),FORM(10)
S,C: 000030010  FORMAT (10A4,I3)
S,C: 000040015  READ(7,10)FORM,I
S,C: 000050020  READ(7,FORM,END=30)(A(J),J=1,I),(B(K),K=1,I)
S,C: 0000600   CALL MATMPY(A,B,C,I,I,I)
S,C: 0000700   WRITE(17,FORM)(C(J),J=1,I)
S,C: 0000800   GO TO 20
S,C: 000090030  STOP
S,C: 0001000   END
SYS:
```

YOU: Y

Even though you selected the card reader, these responses must be entered from the keyboard. The system will automatically revert to the card reader after these prompts.

SYS:

The system informs you that there were no errors.

CIP: P RINT LIST.MAIN20(0),PRTSP=EDIT

SYS:

The system acknowledges your PRINT command and assigns it a batch sequence number (BSN). For this example, assume a BSN of 0137.

CIP: K B

The KB card is encountered, which switches control to the keyboard. The system's underscore prompt for a command is followed by the KB card image printout. This results in another prompt for a command.

S,Y: CANCEL 0137

Since you do not now want to see the listings produced by the compiler, you cancel the task created to print them. Note that LIST.MAIN20(0) is cataloged for you in case you want to print it later.

S,Y: CB

The system prompts you for input of data by unlocking the terminal keyboard. You have the following deck ready in the card reader and switch to card input by entering CB at the keyboard.

Your READ statement refers to data set reference number 7. Since you did not supply a DDEF for FT07F001, you default for terminal input.

CARDS

```
CALL MAIN20
(3F10.6)                                003
+03.600000-04.723000+07.245000
-04.200000+09.000000+03.400000
+05.      - 6.      +07.800000
+35.      +01.300000-04.200000
-07.600000+03.700000+83.0
+11.100000-13.140000+17.8
%END
KB
```

Your program reads its input data from the card deck.

```
CIP: CALL MAIN20
CIP: (3F10.6) 003
CIP: +03.600000-04.723000+07.245000
CIP: -04.200000+09.000000+03.400000
PGM: 25.999969-61.210037 93.89157
```

You default the data set reference number 17 so that your output will be printed at the terminal.

```
CIP: +05.      -6.      +07.800000
CIP: +35.      +01.300000-04.200000
PGM: 54.431976116.639954 44.063965
CIP: -07.600000+03.700000+83.0
CIP: +11.100000-13.140000+17.8
PGM: 64.799973-77.759964101.087952
SYS: CHCIW STOP
S,C: KB
```

Your program concludes. The KB card causes SYSIN to be switched to the keyboard. You then log off.

S,Y: LOGOFF
SYS:

The system confirms your LOGOFF command.

Example 21: Intra-Task Carryovers

In this example, you prepare a deck for nonconversational processing, taking into account carryover of data definitions and COMMON blocks within the task. Your LOGON and LOGOFF commands begin in column 3; all other commands may begin in column one.

CARDS

```
LOGON ADUSERID, , , ADACCT29
```

LOGON command parameters must all be on one card. A task for nonconversational execution must be accurately prepared because most errors will cause premature termination of the task or will make the results useless. Note that you do not enter a password in nonconversational tasks.

```
SECURE (TA=3,9) , (DA=1,2311)
```

The SECURE command is needed in all nonconversational tasks that use private volumes, in order to secure devices on which to mount them. This command requests three 9-track tape drives and one 2311 disk unit. Only two tapes will be used simultaneously; the need for securing three will be explained later in this example. If this command is omitted or an insufficient number of units are reserved, your task will be terminated upon execution of the first DDEF command that requests a private device that is not secured.

```
DDEF MYDISK,VP,MYLIBE,OPTION=JOBLIB
```

Since the data set MYLIBE is cataloged, the only parameters needed are those shown here.

```
DDEF FT01F001,PS,MYDATA0A,UNIT=(TA,9),VOLUME=(,324010),LABEL=(,SL)
```

This DDEF command and the next one include the additional parameters needed for uncataloged data sets. The first command refers to an existing volume. Data set characteristics will be obtained by the system from the label that was created when the data set was first written on your private volume. You specify that standard labels exist. Note the - in card column 72 signifying a continued command on cards. The next card starts in column 16.

```
DDEF FT02F001,PS,MYOUT00B,UNIT=(TA,9),VOLUME=(PRIVATE),-  
DCB=(RECFM=F,LRECL=80),LABEL=(,SL)
```

This DDEF command defines a new data set that is to reside on a private volume. By specifying PRIVATE but omitting the volume serial number on the VOLUME parameter you indicate that you want the operator to provide a tape from the installation pool. With the LABEL parameter, you indicate that you want the system to create standard labels.

```
CALL MAIN21A
```

MAIN21A contains a "READ(1,n)list" statement that reads input from the data set MYDATA0A. It creates the new data set, MYOUT00B, with a "WRITE(2,n)list" statement.

The data set reference numbers 1 and 2, in the READ and WRITE statements, respectively, appear in the data definition names (i.e., FT01F001, FT02F001).

```
UNLOAD MAIN21A
```

Unloading a program once you are through with it is often essential to successful program execution. For example, MAIN21A and MAIN21B both have an unlabeled (blank) COMMON block. You are not sure which COMMON block is larger. If MAIN21B contains a larger COMMON block than MAIN21A, then you may encounter problems when you run MAIN21B because the loader will not allocate virtual storage for a new COMMON block if there is one already loaded from the RUN MAIN21A command. Since the loader allocates each unlabeled COMMON block extra pages of virtual storage, the larger COMMON block in MAIN21B may fit. If it does not, MAIN21B will probably be terminated because of an invalid address reference. Unloading MAIN12 also ensures that the data sets it refers to are closed.

CATALOG MYOUT00B,N

In order to store the volume serial number in the catalog, this command must precede the RELEASE command below. Because this is a new private volume you cannot know the volume serial number in advance. By cataloging you cause the system to record the assigned volume serial number in the catalog. You can then define the data set for a future session by supplying only the minimum parameters in the DDEF command. Although failure to catalog MYOUT00B at this point would not have caused erasure of MYOUT00B, it would have made it inaccessible during the rest of this task. In conversational mode the assigned volume serial number is printed out and can be used in a DDEF command later in the same session.

RELEASE FT01F001

The next program to be run, MAIN21B, will read the data set MYOUT00B with a statement of the form READ(1,n)list. This read statement will also require a DDEF command with a data definition name FT01F001. If another DDEF command named FT01F001 were issued at this time an ambiguity would result, since the data definition FT01F001 would refer to two different data sets. The system would detect the error, reject the command, and print a diagnostic message. Therefore, you issue this RELEASE command to make the data definition name, FT01F001, available for use with MYOUT00B. Releasing the data definition name also releases the data set name and any secured devices it used. The RELEASE command in this example relinquishes one of the secured tape drives to the system, leaving you with two.

DDEF FT01F001,,MYOUT00B,OLD

You are, in effect, moving the data set name MYOUT00B from data definition FT02F001 to FT01F001. There is no ambiguity because the system will remove the data set name MYOUT00B from the data definition FT02F001. The system is also cognizant of the volume in which the data set MYOUT00B is resident and that MYOUT00B has already been mounted on a secured device for this task.

DDEF FT02F001,PS,FINALOC,UNIT=(TA,9),VOLUME=(PRIVATE),-
DCB=(RECFM=FBA,LRECL=50,BLKSIZE=500),LABEL=(,SL)

The FT02F001 data definition was left without a data set name and is thus available for use. This DDEF command now makes use of it to define a new data set named FINALOC. The third secured tape unit (unused until now) will be used to mount the necessary private tape volume.

RUN MAIN21B

A "WRITE(2,n)list" statement in MAIN21B will create the data set FINALOC.

CATALOG FINALOC,N

This command will catalog FINALOC so that you need specify only the minimum parameters when you issue a DDEF command for it in a future session.

LOGOFF

Example 22: Survey of System Facilities and Some Housekeeping Methods

In this example you do some housekeeping, erasing unwanted programs, and transferring a source program from public storage to tape. After logging on, you issue:

S,Y: PC?

This command is used to present the name, access, and, for shared data sets, the owner's identification for one or more cataloged data sets. Since you do not specify any data set names here, information about all your cataloged data sets is presented.

```
SYS: DATA SETS IN CATALOG WITH QUALIFIER ADUSERID
      ADUSERID.JOBLIBA, ACCESS:RW
      ADUSERID.LIST.MAINTTEST(0), ACCESS:RW
      ADUSERID.M22OUT, ACCESS:RO
      ADUSERID.PROJECT.A, ACCESS:RO
      ADUSERID.PROJECT.B, ACCESS:RO
      ADUSERID.PROJECT.C, ACCESS:RW
      ADUSERID.PROJECT.C2, ACCESS:RO
      ADUSERID.SOURCE.MATRIX7, ACCESS:RW
      ADUSERID.SOURCE.TRIALX, ACCESS:RW
      ADUSERID.VERSION5, ACCESS:RO, OWNER:MICHAELB
```

S,Y: DSS? SOURCE.MATRIX7

You want more information about one of your cataloged data sets, so you issue a DSS? command. This command presents more detailed information than does the PC? command. If DSS? is specified without an operand, detailed status information for all your cataloged data sets is presented.

```
SYS: ADUSERID.SOURCE.MATRIX7
      ACCESS: RW
      VOLUME: 232323 (2311)
      DS ORGANIZATION: VI          PAGES: 002
      REFERENCE DATE: 257/68      CHANGE DATE: 257/68
      RECORD FORMAT: V           RECORD LENGTH: 00132
      KEY LENGTH: 00007          RELATIVE KEY POSITION: 00004
```

S,Y: ERASE SOURCE.MATRIX7

S,Y: ERASE LIST.MAINTTEST(0)

You decide to erase two of the foregoing data sets. You would like to examine one other data set before deciding whether to erase it.

S,Y: LINE? SOURCE.TRIALX

SYS: 0000100 C TRIALX IS BASED ON FORMER TUBE TEST ROUTINE.

YOU: (press attention button)

The first line printed out is sufficient for you to recognize this as an old program that you no longer need. You halt further printing by pressing the attention button. You erase it and another data set of numerical data associated with it.

S,Y: ERASE SOURCE.TRIALX

S,Y: ERASE PROJECT

SYS:

The system gives you the option of erasing (E) or retaining (R) the individual data set cataloged under the generic name PROJECT, or erasing all of them (A).

You decide to erase all but PROJECT.B.

SYS: PROJECT.A
YOU: E
SYS: PROJECT.B
YOU: R
SYS: PROJECT.C
YOU: A

By typing A, you cause the system to erase PROJECT.C and those data sets whose names would follow if prompting continued in this case, just PROJECT.C2).

S,Y: ERASE VERSION5

The underscore indicates the end of PROJECT data sets, and you tell the system to dispose of another obsolete data set.

SYS:

The system informs you that VERSION5 is not yours to erase, and ignores your command. VERSION5 is a shared data set for which you do not have unlimited access, which includes the right to erase.

S,Y: DELETE VERSION5

The DELETE will remove only your catalog entry for VERSION5 but will not affect the data set itself or the owner's catalog.

S,Y: POD? USERLIB

Now you request a list of each object module on your USERLIB.

SYS: ZCOX001 START PRESENT POD DUMP
MAIN7
MAIN10
SUBMATRX
END OF PRESENT POD DUMP

S,Y: ERASE USERLIB(MAIN7)
S,Y: ERASE USERLIB(SUBMATRX)

Individual members of partitioned data set can be erased in this way without erasing the entire data set, in this case your USERLIB.

S,Y: POD? JOBLIBA

This command can be used to obtain information about a library or any other VP data set.

SYS: ZCOX001 START PRESENT POD DUMP
PROG14
MAIN12

S,Y: ERASE JOBLIBA

An entire library data set can be erased. In addition to erasing the data set, the ERASE command also removes the catalog entry, thus having the effect for cataloged data sets of ERASE and DELETE.

S,Y: DDEF DD1,PS,TAPE.M22OUT

You define the copy of the data set that will reside on tape.

S,Y: WT M22OUT,TAPE.M22OUT,ERASE=Y

This command requests a nonconversational task to write the data set M22OUT on tape. The data set must be cataloged or defined for the current session. Several optional parameters are defaulted such as blocking factor, line spacing, etc., described fully in the *Command System User's Guide*. ERASE specifies that the data set is to be erased after completion of the WT task.

SYS:

The system informs you that your request has been accepted.
Your request for writing the tape has been accepted and assigned a separate task. The system will also inform you of the number of the tape onto which it was written.
You conclude your task.

S, Y: USAGE

You enter the USAGE command to inquire about the amount of system resources you have used. Two totals are presented: 1) the amount resources used since LOGON, and 2) the total amount of resources used since you were joined.

The following resources are accounted for: permanent storage, temporary storage, direct-access devices, magnetic tapes, printers, card reader-punches, bulk input, bulk output, TSS/360 tasks, total time that your terminal was connected to the system, and CPU time used.

S, Y: LOGOFF
SYS:

The system confirms your LOGOFF command.

Example 23: Generation Data Groups

In this example you create a generation data group (GDG) to store related data sets. Data sets are stored and cataloged in a GDG with a common set of qualifiers, the order the data sets are stored being used to identify each data set. In the three parts of this example, you create a GDG and refer to generations of it by both *relative* and *absolute* references.

Part 1: Creating a GDG

After logging on, you issue:

```
S,Y: DDEF MYDISK,VP,MYLIBE,UNIT=(DA,2311),-  
      VOLUME=(,230001),OPTION=JOBLIB,DISP=OLD
```

You define a job library that resides on a private disk. It contains the object module that you will execute during this task.

SYS:

The system informs you that the task is waiting for volume mounting. You must wait for the operator to mount your disk.

```
S,Y: CATALOG GDG=TESTRSLT,GNO=5
```

This command creates a generation data group catalog entry. All data sets in this group will have the name TESTRSLT as their leftmost qualifier. The second parameter specifies that five data sets are to be retained in one group. When more than the specified number has been reached, the oldest generation of the group is erased.

```
S,Y: DDEF FT01F001,PS,TESTDATA,-  
      UNIT=(TA,9),VOLUME=(,230002),DISP=OLD,LABEL=(,SL)
```

SYS:

The system informs you that the task is waiting for volume mounting. You must wait for the operator to mount your tape.
The data set TESTDATA contains the input to the program, MAIN23, which creates the first generation of your GDG.

```
S,Y: DDEF FT02F001,VS,TESTRSLT(+1)
```

The dsname in this data definition is a relative reference. Zero (0) refers to the most recent member, minus one (-1) to the one just prior to the latest, minus two (-2) to the one before that, etc. Your positive, nonzero reference (+1) indicates a future generation about to be created.

When the system has completed automatically cataloging TESTRSLT(+1), TESTRSLT(+1) will be known as TESTRSLT(0); when a new generation is entered into a GDG, all of the relative generation numbers are reduced by one. If there were a prior generation at this time, it would become (-1).

A data set which was written and cataloged during a previous session can be added to this GDG. You would simply use the CATALOG command to change the data set name to TESTRSLT(+1).

```
S,Y: CALL MAIN23  
SYS: TERMINATED: STOP  
S,Y: LOGOFF
```

SYS:

The system confirms your LOGOFF command.

Part 2: Relative Reference to a GDG Member

After logging on, you issue:

```
S,Y: DDEF MYDISK,VP,MYLIB2,OPTION=JOBLIB,DISP=OLD
SYS:
```

The system informs you that the task is waiting for volume mounting. You must wait for the operator to mount your disk.

You request a different job library, containing a different version of the MAIN23 object module. Although it is on a private volume you need only the four parameters shown because it was cataloged. No indication is seen here of a volume serial number but it is possible that this volume could be the same as the one in Part 1.

```
S,Y: DDEF FT01F001,VS,TESTRSLT(0),DISP=OLD
S,Y: DDEF FT02F001,VS,TESTRSLT(+1)
```

This version of MAIN23 will take its input from the last GDG generation, and then will create the next GDG generation. Thus, the first DDEF command above defines the input of MAIN23 and the second defines the output data set to be stored as a new generation in the GDG established in Part 1.

```
S,Y: CALL MAIN23
SYS: TERMINATED: STOP
S,Y: LOGOFF
SYS:
```

The system confirms your log-off request.

Part 3: Absolute Reference to a GDG Member

After logging on, you issue:

```
S,Y: DDEF MYJOBL,VP,LIBE005,OPTION=JOBLIB,DISP=OLD
```

You request a job library that resides on a public volume. No waiting for volume mounting is necessary. A third version of MAIN23 will be used. This version also receives its input from one of the GDG data sets, defined by the next DDEF command.

```
S,Y: DDEF FT01F001,VS,TESTRSLT.G0002V00,DISP=OLD
```

You refer to the GDG generation created in Part 2 by its absolute generation name. The general form of this name is

qualifier.GxxxxVyy

where xxxx is the absolute generation number and yy the version number. The name of the GDG is the qualifier. The generation number is automatically incremented as new generations are cataloged; the version number is not. If you wish to replace a GDG generation by another version of the same generation, you have to supply the version number by making it part of the absolute name you refer to. The absolute names of the two data sets created in Parts 1 and 2 are TESTRSLT.G0001V00 and TESTRSLT.G0002V00.

```
S,Y: DDEF FT02F001,VS,TESTRSLT(+3)
S,Y: CALL MAIN23
```

You supply a different relative generation number here. It will be added to the last absolute generation number, leaving a gap in the absolute sequence for possible later insertions (i.e., the absolute names TESTRSLT.G0003V00 and TESTRSLT.G0004V00 are omitted).

```
SYS: TERMINATED: STOP
S,Y: LOGOFF
SYS:
```

The system confirms your LOGOFF command.

Example 24: Tape- and Disk Medium Transfers of Virtual Access Method Data Sets

In this example, you copy VAM data sets from one medium to another. Each of the following commands is illustrated: TV (TAPE to VAM), VT (VAM to TAPE), and VV (VAM to VAM). The data sets *to be copied* are assumed to exist, and are cataloged. After logging on, you issue:

S,Y: DDEF DDVTOUT,PS,COPY1,UNIT=(TA,9),VOLUME=(PRIVATE)
You define the tape copy of the data set.

S,Y: VT ORIGIN1,COPY1
Data set ORIGIN1 already exists as a VAM data set. COPY1 is the name assigned to the magnetic tape copy of this data set.

SYS:
When the data set is successfully copied, you will receive a message indicating the names of the input and output data sets, as well as the file sequence numbers and volume serial numbers used.

S,Y: DDEF DDVTOUT,PS,COPY2,UNIT=(TA,9),VOLUME=(PRIVATE),LABEL=(2,SL)
S,Y: VT ORIGIN2
Here the output data set name is defaulted, making necessary the preceding DDEF command. The output data set name will be ADUSERID.TA000001.COPY2, where TA000001 is an arbitrary number to assure uniqueness for the fully qualified data set name.

SYS:
A system message will inform you when the data set is successfully copied. Any failure to copy successfully will result in a diagnostic message and cancellation of the command.

S,Y: TV ADUSERID.TA000001.COPY2,COPYBACK
The data set just produced on a 9-track magnetic tape is copied on direct-access storage in VAM format.

SYS:
A system message will inform you of the success or failure of the copy operation.

S,Y: VV ORIGIN3,COPY3
The data set named ORIGIN3 is copied into public storage and assigned the name COPY3.

SYS:
A system message will inform you of the success or failure of the copy operation.

S,Y: DDEF PRIVDD,VI,COPY4,UNIT=(DA,2311),VOLUME=(,333333)
S,Y: VV ORIGIN4,COPY4
You desire to copy the data set ORIGIN4 onto a private VAM volume #333333 and name the output data set COPY4.

SYS:
A system message will inform you of the success or failure of the copy operation.

S,Y: LOGOFF
SYS:
You want to copy the data set ORIGIN4 onto a private VAM volume #333333 and The system accepts your LOGOFF request.

Example 25: The Text Editor Facility

In this example, you use the Text Editor to create and edit data sets. The example illustrates only the more basic features of the Text Editor facilities. After logging on, you issue:

S,Y: DEFAULT REGSIZE=8

You enter a default value (in this example, 8 bytes) for the length you wish to allow for the names of regions within the data set you will edit.

S,Y: EDIT EX25

You invoke the Text Editor and specify that data set EX25 is to be processed. Since no DDEF command was issued for EX25, and the value of the REGSIZE parameter is greater than zero, the system assumes that you wish to edit a region data set with the following data set attributes: virtual index sequential data set organization, format-V records, maximum logical record length of 256 bytes, a key length of 15 bytes (8-byte region name and 7-byte line number), and a relative key position of 4.

S,Y: REGION REGION1

You specify a region name; you will be invited to enter text for the region indicated.

S,Y: 0000100 LINEONEE

S,Y: 0000200 LINETWO

S,Y: 0000300 LINETHREE

You enter the data lines you wish to be part of the region named REGION1. Each time you press the return key, the Text Editor prompts with the next line number.

S,Y: 0000400 _UPDATE

You decide to make a change to the previous entries. By preceding UPDATE with an underscore character (_), the Text Editor immediately executes the command.

SYS: FOR EACH LINE ENTER LINE NUMBER AND DATA

YOU: 0000150 INSERT1

You add line number 150 to your data set.

YOU: _INSERT 0000400

You now want to continue entering data at the point where you left off earlier. You precede INSERT with an underscore character, since the system expects data and not a command following UPDATE.

S,Y: 0000400 LINEFOUR

S,Y: 0000500 LINEFIVE

S,Y: 0000600 _END

You terminate Text Editor processing. Since EX26 resides in public storage, and is VAM organized, it will be automatically cataloged.

S,Y: EDIT EX25

S,Y: _EXCISE 200

Line number 200 from the current region REGION1 will be deleted.

S,Y: INSERT 260,10

S,Y: 0000260 INSERT2

S,Y: 0000270 INSERT3

S,Y: 0000280 _END

Text Editor processing is terminated.

S,Y: LOGOFF

You decide to terminate your conversational task.

SYS:

The system accepts the LOGOFF request.

Example 26: The Text Editor Facility

In this example, you make more extended use of the updating capabilities of the Text Editor. The example is probably more complex than you might expect for a single terminal session, but it will serve to portray the scope and flexibility of the editing commands that are available to you. After logging on, you issue:

S,Y: DEFAULT REGSIZE=8

You enter a default value (in this example, 8 bytes) for the length you wish to allow for the names of regions within the data set you will edit.

S,Y: EDIT EX26

You invoke the Text Editor and specify that data set EX26 is to be processed. Since no DDEF command was issued for EX26, the same data set attributes are assumed as for EX25 in the previous example.

S,Y: REGION REGION2

You define a region name for EX26.

S,Y: 0000100 LINEA
S,Y: 0000200 LINEB
S,Y: 0000300 LINEC
S,Y: 0000400 LINED
S,Y: 0000500 LINEE
S,Y: 0000600 LINEF
S,Y: 0000700 LINEG
S,Y: 0000800 LINEH
S,Y: 0000900 LINEI
S,Y: 0001000 LINEJ
S,Y: 0001100 _END

You enter the lines for REGION2 of data set EX26. You then terminate Text Editor processing. Since EX26 is VAM organized and resides in public storage, it is automatically cataloged when the Text Editor is invoked.

S,Y: EDIT EX25

You again invoke the Text Editor and specify that data set EX25, region REGION1, prepared in the previous example, is to be processed.

S,Y: REGION REGION1

The system prompts you to enter a command.

S,Y: NUMBER300,500,BASE=300,INCR=50

Lines 300, 400, and 500 are now numbered 300, 350, and 400.

S,Y: DISABLE

This optional command makes the following data set changes provisional. These changes can be made permanent by issuing an ENABLE command, after you are satisfied all changes were entered properly.

S,Y: EXCERPT EX26,REGION2,600,1000

You wish to excerpt lines 600 to 1000 from REGION2 of data set and insert these lines in the current data set EX25.

S,Y: CONTEXT ,LAST,STRING1=LINE,STRING2=LINE NUMBER

The current region is searched for all occurrences of the character string line. Wherever it is found, LINE NUMBER will replace the occurrence.

Note: This facility is useful for symbol replacement in source language data sets.

S,Y: ENABLE

Up to this point, the revisions made since DISABLE was issued above were temporary. With the execution of the ENABLE command, these revisions are now permanent. A STET command, on the other hand, would have deleted all changes made since the DISABLE was issued.

S,Y: CORRECT N1=100,SCOL=0

You now want to remove a single character from a line, starting with the first column (relative column \emptyset). Standard correction characters are assumed, by default.

SYS: LINEONEE
YOU: * %

The result will be LINEONE.

S,Y: EEDIT EX26

You reopen the text editor for data set EX26.

S,Y: REGION REGION2

The current region is now REGION2.

S,Y: LOCATE \emptyset ,LAST,LINEF

You want the entire data set EX26 to be searched for the character string LINEF.

SYS: 0000600 LINEF

The line in which LINEF is first discovered is displayed at the terminal.

S,Y: LIST100,LAST

The current region (REGION2) will be displayed.

S,Y: END

You terminate Text Editor processing.

S,Y: LOGOFF
SYS:

The system accepts your LOGOFF command.

Example 27: Use of Procedure Definition (PROCDEF)

In this example, you create a procedure, tailored to your needs, to be called at a later time just as if it were a system-supplied command. After logging on, you issue:

```
S,Y: PROCDEF FTNPGM
S,Y: 0000100 PARAM MODULE
S,Y: 0000200 FTN MODULE
S,Y: 0000300 PRINT LIST.MODULE(0),,,EDIT,ERASE
S,Y: 0000400 _END
```

You have defined a procedure which will now be available for calling by the name FTNPGM. It allows you to define a module name for compilation. By calling the established procedure, and giving a unique module name to use, both the compilation, and printing of the resulting listing data sets can be accomplished.

```
S,Y: FTNPGM MYMOD
```

The procedure established above (via PROCDEF) will now be activated. The actual module name (MYMOD) will replace the dummy module name (MODULE) wherever it occurs.

```
S,Y: PROCDEF SETUP
S,Y: 0000100 PARAM STORED=$1, ISD=$2, SLIST=$3, CRLIST=$4
S,Y: 0000200 DEFAULT STORED=$1, ISD=$2, SLIST=$3, CRLIST=$4
S,Y: 0000300 _END
```

This procedure will now be available to vary the default values for certain FTN parameters.

```
S,Y: SETUP Y,N,Y,Y
```

Some FTN parameter default values have been adjusted to suit your requirements.

```
S,Y: FTN MOD1
.
.
.
```

You now proceed with the compilation of MOD1 with the adjusted default values.

```
S,Y: PROCDEF ZLOGON
```

Here the operand (ZLOGON) for the PROCDEF command is shown without the use of a keyword.

```
S,Y: 0000100 DDEF STOREIT,VP,MYLIB,OPTION=JOBLIB
S,Y: 0000200 _END
```

You decide that each time you LOGON you would like a certain job library defined for any object modules you may produce. By assigning ZLOGON as the procedure name, you insure its automatic call as soon as LOGON is accepted. MYLIB is assumed to be an existing cataloged data set. Since PARAM was **not** used you **cannot change** any of the values in the DDEF command.

```
S,Y: LOGOFF
SYS:
```

The LOGOFF command is accepted by the system.

Example 28: The User Profile Facility

In this example, you are shown how to manipulate your copy of the prototype user profile, made available to you at JOIN time. This prototype profile is a member of your user library (USERLIB). After logging on, you issue:

S,Y: DEFAULT DSORG=VS

The data set organization field was originally defaulted by the system to VI (index sequential). You will now be using mostly VS organized data sets, so you set the default value (for the DDEF command) to virtual sequential (VS).

S,Y: SYNONYM DOPROG=FTNPGM

The FTNPGM procedure named in Example 27 can now be invoked with either name: DOPROG or FTNPGM.

S,Y: SYNONYM FINIS=DISPLAY 'TASK COMPLETED'

When FINIS is invoked, the message: TASK COMPLETED will appear on SYSOUT.

S,Y: PROFILE

The above changes apply to your session profile only. You now decide to make the changes a permanent part of your user profile.

S,Y: FINIS
SYS: =TASK COMPLETED

Since the PROFILE command was invoked, FINIS may be used in all subsequent sessions to produce the same message.

S,Y: LOGOFF
SYS:

The system accepts your LOGOFF command.

The appendixes give detailed information on the use of TSS/360 by the FORTRAN programmer; they contain the following information:

- A. Use of the FORTRAN Compiler: describes source statement entry and correction and compiler diagnostics, output, and restrictions.
- B. PCS and FORTRAN Object Programs: describes the use of PCS when debugging FORTRAN object programs at the source language level.
- C. Programming Considerations: describes the programming techniques that yield more efficient programs, the effect of compiler optimization on the use of PCS, system naming conventions, library management, and several miscellaneous considerations.
- D. Assembler Language Subprograms: describes the techniques and conventions of incorporating assembler-language subprograms into FORTRAN language programs.
- E. Specification of Data Set Characteristics: describes the various record formats and data set organizations available to the FORTRAN user and explains how and when to specify these characteristics for a data set.
- F. Attention Considerations: describes the system's response to the attention key, which is a function of the system's current activity.
- G. Command Formats: describes the notation used to present commands and gives the general form of each of the TSS/360 users' commands.
- H. Carriage and Punch Controls: lists the extended ANSI FORTRAN carriage control and punch control standard characters.
- I. Sample Program: describes the sample program that is distributed with the TSS/360 FORTRAN IV compiler.

Appendix A. Use of the FORTRAN Compiler

This appendix discusses the following topics:

1. Entry and correction of FORTRAN source statements: the format of keyboard and card source statements, efficient source statement correction techniques, and entering lines from the keyboard that can later be punched and reentered as cards.
2. Compiler diagnostic action: the format and effect of diagnostic messages and error severity codes produced by the compiler.
3. Compiler options and listings produced: FTN parameters, listing options, indirect references to FORTRAN IV subprograms, and the destinations of all compiler output.
4. Compiler restrictions: simple and complex source program restrictions.

Entry and Correction of FORTRAN Source Statements

This section discusses the following topics:

1. The format of FORTRAN source statements entered at the terminal keyboard, at the terminal card reader, and at a card reader at the central computing facility.
2. Guidelines for efficient source statement correction techniques.
3. Techniques for entering FORTRAN source keyboard lines so that they can be punched and reentered in card form for compilation.

Format of Source Lines

A FORTRAN source *statement* is composed of one or more individual source *lines*. A source line consists of a single card or a single line of keyboard input. There are five types of lines:

1. Comment lines
2. Initial lines
3. Continued lines
4. Continuation lines
5. END lines

The following paragraphs describe each of these types of lines for both card and keyboard format.

For both card and keyboard statements, the maximum number of text characters that can be contained in a statement is 1320. Card input lines normally con-

tain 66 text columns per card (columns 7-72); thus, an initial line and 19 continuation lines are allowed. The number of keyboard lines contained in 1320 characters depends upon the position of the first character in each line and the last character given prior to issuing a carrier return in each line.

Card Format Line (Both Nonconversational and Console Card Reader)

Comment lines are those lines with the letter C in column 1. The compiler ignores these lines, but includes them in the source listing. Comment lines may appear anywhere within the source program except immediately preceding a continuation line. *Initial* lines in card format are lines with the characters zero or blank in card column 6. An initial line is the first line of a FORTRAN statement. *Continued* lines are lines whose content does not complete the statement; that is, the statement is continued on the following line. In card format, a line is not known to be a continued line until the following line is inspected. A *continuation* line is a line in a statement other than the initial line, that is, a line whose content is a continuation of the preceding line. In card format, a continuation line must contain a character other than zero or blank in card column 6. An *END* line is a line containing the FORTRAN END statement. Every source module must terminate with an END line.

Character Sets — Card Format

The CA and CB commands transfer control to the 1056 card reader, and specify the character set to be used. CA is used to convert card input from 1057 card punch code to EBCDIC. CB specifies conversion from 029 key-punch code to EBCDIC.

Keyboard Format

Keyboard lines are different from card lines in two general ways: (1) the rigid column format of card lines is relaxed in keyboard format and (2) the maximum number of characters in a line is greater (as many as 130, depending upon the type of keyboard being used).

Keyboard *comment* lines are similar to card comment lines in that a keyboard comment line is identified by the letter C in the first typeable position, and may appear anywhere within the source program except immediately preceding a continuation line. Note that the statement C = 1.0, for example, if begun

in type position 1, would be treated as a comment line. A good general practice is to set tab stops and make use of the tab key. This practice will (1) ensure that a statement beginning with the letter C will not inadvertently be processed by the compiler as a comment line, and (2) provide a simple method for formatting the input program on the terminal paper — separating statement numbers from statement text, etc.; most examples in this appendix imply or expressly describe use of tabs. The compiler considers a tab the equivalent of *one* blank, when entering keyboard lines. Thus, when reference is made to blanks, tab *or* blank is implied, unless specifically stated otherwise.

Initial lines in keyboard format can contain a statement number, as in card format. This statement number can begin in any type position (hereafter referred to as position) and is distinguished from the text portion of the line by being numeric rather than having an alphabetic character as the first character. Statement numbers can be preceded by any number of blanks or tabs. A statement number can contain from one to five numeric characters, each of which can be preceded or followed by any number of blanks. A statement number is terminated when either five numeric characters have been encountered or a non-numeric, nonblank character is encountered, whichever occurs first. A statement number cannot be contained on more than one line. Following the statement number is the line text. The text must be separated from the statement number by at least one blank, or by the character zero. (The user is warned that the entry of a terminal line with a statement number of 100, for example, immediately followed by the text of the statement — i.e., with no intervening blank — causes the second zero of the statement number to be treated as the character separating the statement number and the text, and the statement number will be considered by the compiler to be 10, not 100.) The text portion of a line is terminated either by a hyphen (-) followed by a carrier return (if the line is to be continued), or by a return preceded by a character other than a hyphen (if the line is not to be continued).

A keyboard *continued* line is one ending with a hyphen and a carrier return. The hyphen, of course, is not considered part of the text.

A keyboard *continuation* line is the line following a continued line. A continuation line cannot have a statement number. The text portion of a continuation line can begin in any position and can be preceded by any number of blanks; it must also be preceded by a single character other than blank or zero. This single character is equivalent to the column 6 continuation character of card lines and thus is termed a

“floating continuation character.” In effect, the first nonblank, nonzero character of a continuation line is treated as a continuation character. The floating continuation character, which is not considered part of the text, is useful for two reasons:

1. It allows the user to tab the continuation line (so that it will fall directly beneath the preceding line) without thereby causing a blank to be inserted in his text; blanks or tabs are ignored in continuation lines until the floating continuation character is encountered. This is an important consideration, for example, in cases where a line break must occur in the middle of a character string that does not permit embedded blanks.
2. If lines entered via the console are to be punched and later reentered in card format, the floating continuation character makes this possible (this is described later in this appendix).

A keyboard END line is a line containing the FORTRAN END statement. Every source module must terminate with an END line.

Character Sets — Keyboard Format

KA and KB are used to specify the character set to be used during keyboard input. KA specifies the full EBCDIC character set during input. KB indicates that the lower-case characters (a-z and !”¢) be translated into their upper-case equivalents (A-Z and \$ # @ respectively).

It should be noted that KA is in effect only for the command during which it is issued. The system underscore (inviting a new command) reverts to the KB folded mode.

Mixed Card and Keyboard Input

FORTRAN source statements entered at the terminal may be from cards, from the keyboard, or from a mixture of the two. If the first line of a statement is from a card, all lines in that statement must also be on cards. If the first line in the statement is from the keyboard, the following lines may be from the keyboard or from cards. Once the source of a statement becomes cards, however, keyboard lines may no longer be included in that statement.

Once a line is entered at the terminal in either card or keyboard form, the source of the line (card or keyboard) is retained even though the line may be stored in a prestored data set and later presented to the compiler from that source. Thus the above mixed-input rules apply for both terminal input and later corrections to that input.

The procedure for changing sources is as follows. Keyboard lines will be expected by the system until

C, CA, or CB is entered at the keyboard. Once these characters have been entered, input lines are expected to be on cards. If a card containing K, KA, or KB is encountered, lines are once again expected from the keyboard. (See *Terminal User's Guide* for SYSIN device selection and data translation).

Examples: In these examples, the following notes apply:

1. The letter t represents depression of the tab key. The keyboard position at which the carrier is to be positioned as the result of a tab is implied by the position of the next nonblank character in the example. The following examples assume tab stops are set such that the first depression of the tab key causes the carrier to be positioned at type position 6 and the next depression causes positioning at type position 7.

EXAMPLE	FORMAT	CARD OR TYPE POSITION
		1 567
1	Card C	COMMENT
NOTE: Comment line.		
2	Card 50	Y=1.0
NOTE: Initial line with a statement number.		
3	Card	Y=2.0
NOTE: Initial and continued line, no statement number.		
	Card	X+B
NOTE: Continuation line.		
4	Key C	COMMENT
NOTE: Comment line.		
5	Key 9	Y=3.1416
NOTE: Initial line with a statement number.		
6	Key 100t tZ=1.0-(CR)	
NOTE: Initial and continued line; tab to column 6, tab again to skip the continuation character position, then at position 7 type first text character.		
	Key t	X+3.0*B
NOTE: Continuation line; tab to position 6 for floating continuation character, with text beginning in column 7.		
7	Key 200t tY=1.0-(CR)	
NOTE: Initial and continued keyboard line.		
	Key CB	
NOTE: To note a card line follows.		
	Card	X+3.0*B
NOTE: Continuation card line.		

2. The characters (CR) represent a carrier return.
3. The character (-) is used as the continuation character by system default.

Efficient Correction Techniques

Conversational correction of FORTRAN statements is normally made at one of two points in the compilation. The first, termed here *local* correction, occurs following the compiler's scan of the statement just entered and its printing at the terminal of any diagnostic messages associated with that statement. The second point at which corrections are normally made occurs when the entire program has been entered (i.e., when the END statement has been entered) and a message similar to, "MODIFICATIONS? ENTER Y OR N" has been typed at the terminal. Corrections made at this point are termed here *global* corrections. The distinction between global and local corrections (and between different types of local corrections) is important in that the user can minimize the amount of processing required for a given compilation by being aware of the effect corrections have on the compilation process. The following paragraphs describe efficient correction techniques in detail. Before proceeding to this detailed discussion, however, a general correction rule that applies in *most* cases can be given: the most efficient correction technique is to correct erroneous statements immediately after they are entered and the diagnostic is produced. This is the most natural time to correct an erroneous statement, and is a convenient guideline to use.

After all *global* corrections have been made, the entire source program is rescanned, beginning with the first source line of the program. When *local* corrections are made, reinitiation of the source scan may or may not be required, depending upon the type of correction made. It is desirable, of course, to minimize the number of source scans. For this reason, corrections noted below as reinitiating the source scan generally should *not* be made until the time for global corrections is reached. This rule does not apply where failure to make a correction would result in many other diagnostics, such as an error in entering variables in a DIMENSION list.

Discussion of compiler response to various types of local corrections can be quite complex if all possible combinations of conversational card and keyboard corrections are covered. This section does not discuss all possible types of corrections specifically, but gives sufficient descriptive material and examples so that the compiler response in any given case can be determined.

Discussion of compiler response to local corrections requires a definition of three terms:

1. Partial statement — a statement currently being entered. For card input, a statement is partial until a noncontinuation line is entered to terminate the statement. Keyboard statements are partial until a line is entered that is not a continued line or a comment line. An example of a partial keyboard statement is the following (i.e., the line is noted as being a continued line, but the continuation line has not yet been entered):

$$X=A+B-(CR)$$

2. Tentative statement — the last *complete* statement entered to the compiler. A keyboard statement becomes tentative when the last line of the statement is not a continued line, causing the line to be scanned by the compiler. This statement will remain tentative until a new statement is completely entered. Thus a new statement may be begun without changing the tentative status of the previous statement, but once the new statement has been completely entered, the statement held in tentative status becomes committed (see below). Entry of comment lines does not affect the status of tentative statements. Such lines are included in the listing but otherwise ignored.
3. Committed statement — the statement preceding a tentative statement. The relation between a committed statement and a tentative statement is identical to the relation between a tentative statement and a partial statement: once a statement becomes tentative, the preceding statement becomes committed. In the following example, entrance of the (non-continued) second statement causes the second statement to become tentative and causes the first statement to become committed. Both are keyboard lines.

$$X=A+B(CR)$$

$$Y=X**2(CR)$$

Entry of comment lines does not affect the status of tentative or committed statements — such lines are included in the listing but otherwise ignored.

The relation between the above types of statements and compiler response to corrections is as follows:

1. Tentative and partial statements can be corrected without causing a reinitiation of the source scan.
2. Correction of committed statements causes reinitiation of the source scan from the beginning of the program.
3. Insertion of a new statement between a committed and a tentative statement does not cause a rescan

of the entire program. Such an insertion causes a rescan of the tentative statement, however.

The above rules can be more easily understood by inspection of the following examples. In these examples, line numbers are seven digits with the last two digits zero, the format in which they are printed by the system. A line number underneath a number sign, #, is a correction line. Such line numbers need not be seven digits long and need not have two trailing zeros. This notation is consistent with the system convention by which printing of a diagnostic is followed by printing a number sign soliciting a correction by the user. The correction line numbers are followed by a comma, also consistent with system conventions. Line numbers preceded by a percent sign (%), where the % is entered by the user, are unsolicited corrections — that is, corrections originated by the user and not arising from the system's diagnostic action.

Conversational corrections also affect the source listing, as follows:

1. Source statements and their associated diagnostics are not added to the source listing until the statements are committed. Thus, the terminal listing may contain many diagnostic messages, but the source listing contains only those diagnostics not corrected.
2. Corrections causing reinitiation of the source scan cause the source listing plus diagnostics associated with the previous source scan to be lost, since the results of the new source scan replace those of the previous scan.

Example 1.

LINE NO.	FORMAT	CARD OR TYPE POSITION
		1 567

0000200	Key	Z=1.0
0000400	Key	Y=2.0
0000600	Key	DO 10 I=1,10

NOTE: In this example, all lines begin in type position 1, a dangerous practice as any lines beginning with a C are considered comment lines.

0000800	Key	X(I)=I
---------	-----	--------

NOTE: Causes a diagnostic, as no DIMENSION (or equivalent) statement is included.

#		
100,	Key	DIMENSION X(10)

NOTE: Making a correction prior to the committed statement (line 600) causes reinitiation of the source scan. Note that this statement could have been inserted at 700 rather than 100 without causing reinitiation of the source scan.

Example 2.

LINE NO.	FORMAT	CARD OR TYPE POSITION
		1 567
0000100	Key t	tZ=10.0 (CR)
0000200	Key t	tY=3.0- (CR)
0000300	Key t	X=B- (CR)
0000400	Key t	X+C- (CR)

NOTES:

The t indicates the tab key has been depressed, as discussed earlier.

The user is requested to enter line 500, but he realizes that line 300 must be corrected. When the correction of line 300 is made, the statement starting at line 200 is in partial status. This correction does not cause reinitiation of the source scan.

0000500%300, Key t X-B- (CR)

Note the use of floating continuation characters in lines 300 and 400.

0000500 Key t X+D (CR)

NOTE: Entry of line 500 completes the statement. The compiler then processes the composite statement $Y=3.0-B+C+D$.

Example 3.

LINE NO.	FORMAT	CARD OR TYPE POSITION
		1 567
.		
.		
.		
0000400	Key	X=1 (CR)
0000500	Key	B=2 (CR)
0000600%400,	Key	X=2 (CR)

NOTE: The user is requested to enter line 600. When line 500 was entered, the statement of line 400 became committed. The correction of line 400 thus causes reinitiation of the source scan.

Entry of Keyboard Source Statements for Later Punching and Recompile

The entry of source statements such that they can be later punched out and reentered in card format is governed by the following considerations:

1. Source lines reside in a line data set in a format in which the initial input source line is *preceded* by eight characters — a 7-byte zoned-decimal key and a character specifying to TSS/360 that the source line was entered in card form or at the terminal keyboard.
2. A continued line (hyphen preceding the carriage return) when punched and reentered in card format retains the hyphen unless precautions are taken to remove it (see below).
3. Keyboard input positioning requirements are much more flexible than for card input.

Therefore:

1. Keyboard input lines must contain all statement numbers in columns 1-5.
2. Keyboard input lines must contain no tabs.
3. Keyboard input lines must contain all floating continuation characters in position 6.
4. Text must be contained between position 7 and a position not greater than 72, as source *card* lines cannot be scanned beyond column 72 for characters to be included in the text.
5. Continued lines must all contain the hyphen continue character in a column such that the PUNCH command (see below) will cause the hyphen *not* to be punched.
6. The column chosen in the PUNCH command for termination of punching must be such that no lines contain *text* beyond this column.

If input lines are prepared in accordance with the above rules, the line data set can then be punched, using the PUNCH command. The parameters supplied in the PUNCH command are: startno, the position in the source line as it resides in the line data set that will be punched as column 1 of the card; and endno, the position in the source line that will be last punched in the card. If the program contains no continuation lines (one line per source statement), good choices for startno and endno would be 9 (to skip 8 bytes — the 7-byte line number and the keyboard character) to 88 (to punch up to 80 columns of text from each source line — this choice assumes no source lines contain more than 80 columns of text).

The selection of 9 for startno will nearly always be the proper choice. The endno selection will vary, of course, depending upon position of text and continuation characters in the program source lines.

Compiler Diagnostic Action

This section describes the format of all diagnostic messages produced by the TSS/360 FORTRAN compiler. It includes a description of the error severity code and error level associated with each message, and describes the effect of error severity upon requests to execute the compiled program. Refer to *System Messages* for a description of each diagnostic and the source program errors that cause it.

The FORTRAN compiler issues diagnostic messages for source program errors, for violations of compiler space and size restrictions, and for apparent machine errors. These messages are included in the source program listing produced by the compiler if a listing

is requested; they are also printed at the terminal in conversational mode. (A detailed description of the destination of diagnostic messages for all combinations of input sources, options requested, etc., is given in Table 3 later in this appendix.)

Nearly all compiler diagnostic messages are printed on one line. The format of this line is:

number code *** text

(In the two-line messages, the second line follows the first and omits the number and code fields.)

The "number" parameter is the source program line number (*not a FORTRAN statement number*) of the first line of the statement to which the message applies. Messages concerning errors that the compiler does not associate with any specific statement carry the line number of the source program END statement.

The "code" parameter is a one-letter indicator of the severity of the error and the action taken by the compiler. The letters used, the severity of errors associated with each letter, and a brief description of compiler action taken are given in Table 2.

When a FORTRAN main program is to be executed using the CALL command, the module named in the CALL command and all modules called by this module are inspected during the loading process to see whether any have been compiled with level-2 errors (severity codes E or F). Any module containing an error level of 2 causes a diagnostic message naming the module and the error level to be printed on the user's SYSOUT.

The "text" parameter in diagnostic messages is a verbal description of the condition that caused the message. Names, statement numbers, etc., from the FORTRAN source program being compiled are included, where applicable. In addition to information to locate and identify the condition, often "text" specifies the actions taken by the compiler in F-code messages. Occasionally the object code listing must be inspected to determine the actual compiler action.

Occasionally, two or more identical messages are produced for a source program statement in which the erroneous situation appears to occur only once. This is due to the conjunction of two conditions: (1) the error is not serious enough to force the compiler to abandon processing the statement and (2) the statement is of a kind that requires more than one scan by the compiler. Examples of such conditions are (1) when an argument of a statement function is in error and the argument is used more than once in the statement function definition and (2) when certain types of errors are made in an I/O statement list.

The compiler does not attempt to make systematic checks for machine failure. However, in the course of processing, it may test for conditions that ought never to arise, according to the design of the compiler. If any such condition is detected, an A-code message is issued advising of machine or compiler error. These messages should be brought to the attention of system maintenance personnel.

Although the conversational user may know that a certain diagnostic is to appear at a particular place in

Table 2. Compiler Diagnostic Action

CODE	SEVERITY	DESCRIPTION	ACTION
W	Warning, Level-1 error	The message is a warning that the compiler has detected either a situation that may not be as the programmer intended or a use of a language feature that is acceptable to the TSS/360 FORTRAN compiler but is unacceptable to other implementations of IBM System/360 FORTRAN. In either case, the statement is compiled exactly as written.	Statement compiled as written
E	Serious, Level-2 error	The message is a notification that the compiler has detected a serious source program error. Compilation continues, completely ignoring the offending statement, and the object program is generated as if the statement had not occurred. (Although the offending statement is deleted from the compilation, the program is marked as containing a serious error if the offending statement is not replaced. The presence of a serious error causes a diagnostic message to be printed on the user's SYSOUT data set at load time.	Statement deleted
F	Serious, Level-2 error	This message is also a notification that the compiler has detected a serious error. The compilation continues by partially compiling the statement. The effect of F-code errors on the generated object program can frequently be determined from the text of the message, but occasionally examination of the object code listing is required. Object program execution is not recommended.	Statement partially compiled
A	Serious, Level-3 error	The message is a notification of a situation serious enough to prevent continuing the compilation. After issuing an A-code message, the compiler exits to the command-language level. These messages are concerned with violation of compiler size or space restrictions.	Compilation discontinued; no object module produced

his program, or he may recognize a diagnostic as it begins to appear at the terminal, he should not try to save time by pressing the attention key to prevent full diagnostic text printout. Little time can be saved, since a RUN command must be entered after an attention, and much time may be lost, since the compiler will reinitiate compilation with the first source line if certain of its diagnostics are so interrupted.

Compiler Options and Listings Produced

This section discusses three topics:

1. The parameters that must be supplied to the compiler when the FTN command is given.
2. The listings produced by the compiler when requested by user-supplied parameters.
3. A list of FORTRAN IV supplied subprograms which can be called by the compiler as a result of exponentiation, interrupt handling, I/O processing, or STOP or PAUSE statement usage.
4. The destination of all output from the compilation.

FORTRAN Parameters

After issuing a FTN command, the user must enter a parameter providing the module name for this compilation. A list of compiler parameters is given in Figure 7, and explained in detail following the figure. The notation used in Figure 7 is explained in Appendix G, "Command Formats."

Some of the compiler parameters listed in Figure 7 must be provided by the user; others may be left unspecified and default values will be chosen. In some cases the user will be prompted for missing parameters; in other cases he will not. Figure 8 shows the relation between parameter specification and the compiler conditions. In Figure 8 the terms "explicitly defaulted" and "implicitly defaulted" are used as discussed below:

Explicitly Defaulted

A comma is issued immediately following entrance of the preceding parameter, rather than entering a value for the new parameter followed by a comma. For example, module ALPHA, with prestored source lines, is to be compiled, explicitly defaulting the version identification, but supplying values for all other parameters. The proper parameter description is:

ALPHA, Y,,Y,Y,Y,Y,Y,Y,Y

Implicitly Defaulted

In the example of the preceding paragraph, the user could depress the return key following entrance of the Y specifying an ISD is to be produced. This action causes all parameters following the ISD option to be implicitly defaulted.

The parameters shown in Figure 7 are described as follows:

NAME — specifies the name of the object module to be created. Prior to selecting the module name, the user should refer to Appendix C for a discussion of TSS/360 naming conventions, in order that a module name not be chosen using a reserved system name. The source data set for the module is named by appending a period and the module name to the characters SOURCE. For example, module ALPHA will have a source data set name SOURCE.ALPHA. The module name must be unique to the library that includes it (i.e., it must not be the same as any entry point, control section name, or module name in that library). Similarly, the module name must not be the same as any COMMON block, FUNCTION, SUBROUTINE or any variable name used within the program being compiled. The name consists of one to eight alphameric characters, the first of which must be alphabetic. It is recommended that module names be six or fewer characters, as discussed in Appendix C.

Default: None; a module name must be specified.

OPERATION	OPERAND
FTN	NAME=module name [, STORED= {Y} {N}] [, VERID=version identification] [, ISD={Y N}] [, SLIST={Y N}] [, OBLIST={Y N}] [, CRLIST={Y N}] [, STEDIT = {Y N}] [, MMAP={Y N}] [, BCD={Y N}] [, PUBLIC={Y N}] [, LISTDS={Y N}] [,LINCR=(first line number, increment)]

Figure 7. FORTRAN Parameters

Parameter	User Action	System Action	
		Nonconversational Mode	Conversational Mode
Module	Explicitly or implicitly defaulted	Task terminated	Prompting for parameter will occur
	Not defaulted	Processing continues with user-supplied parameter	
All other options	Explicitly defaulted	Default values selected	
	Implicitly defaulted	Default values selected	
	Not defaulted	Processing continues with user-supplied parameter	

Figure 8. Compiler Parameters Default and Prompting Description

STORED — specifies whether or not the source data set is prestored (if so, it must have been named SOURCE.module). The allowable values are Y or N.

Default: N.

VERID — specifies the version identification to be assigned. The version identification consists of one to eight alphameric characters.

Default: If a version identification is not assigned, the version of the object module may be determined by using the system supplied "time stamp." A time stamp is always produced by the system; it gives the current time and date at which the compilation begins. This time will be different from the time at which any other compilation begins, thus allowing positive identification of the compilation output. The compilation listing will contain an edit of the time stamp in the following format: MO/DD/YY HH:MM:SS giving the time of the compilation in month, day, year, hour, minute, and second.

ISD — specifies whether an internal symbol dictionary, which is required for extended use of PCS, is to be produced. The allowable values are Y or N.

An ISD should not be requested unless PCS is to be used with this compilation since the request for an ISD also inhibits compiler optimization. See Appendix C for a full discussion of this topic.

Default: Y.

SLIST — specifies whether a source program listing is to be produced. The allowable values are Y or N.

Default: Y.

OBLIST — specifies whether an object program listing is to be produced. The allowable values are Y or N.

Default: N.

CRLIST — specifies whether a cross reference listing is to be produced. The allowable values are Y or N.

Default: N.

STEDIT — specifies whether the symbol table is to be listed. The allowable values are Y or N.

Default: N.

MMAP — specifies whether a storage map is to be produced. The allowable values are Y or N.

Default: N.

BCD – specifies whether input includes BCD (binary-coded decimal) or EBCDIC form of special characters. The allowable values are Y for BCD or N. If Y (BCD) is chosen, either BCD or EBCDIC input may be used. Thus EBCDIC corrections may be made to BCD source lines. This option is included for compatibility with FORTRAN programs written for previous IBM systems. If the user is uncertain as to whether his program uses the BCD or EBCDIC form of special characters, the user should choose the Y option, thereby giving himself complete flexibility.

Note: If the EBCDIC option is selected, statement numbers passed as arguments must be preceded by an asterisk (&n). However, if the BCD option is selected, statement numbers passed as arguments must be preceded by a dollar sign (\$n), and the \$ character must not be used as an alphabetic character in the source module. (n represents the statement number.)

Default: N.

PUBLIC – specifies whether the executable part of the module (the CSECT, described below) to be created is to have a public or private attribute. Allowable values are Y or N; Y indicates public; N indicates private. If the public attribute is chosen, other programmers may use the same program if they are also given access to the library in which this module is stored. The means for so doing are described in Appendix C.

Default: N.

LISTDS – specifies whether the requested listings are to be placed in the list data set or on SYSOUT. Allowable values are Y or N. If Y is chosen, the listings are placed in the list data set and can be printed at any time with the PRINT command. If N is chosen, the listings are placed on SYSOUT; in other words, it is printed automatically but not kept in the system.

Default in conversational mode: Y.

Default in nonconversational mode: N.

LINCR(line) – specifies the line number to be assigned to the first line of the data set. The line number contains three to seven digits, of which the last two must be 00.

Default: 100.

LINCR(increment) – specifies the increment to be applied to develop successive line numbers. The increment may contain three to seven digits, of which the last two must be 00. It cannot be negative.

Default: 100.

Example: A user specifies the following parameters when issuing the FTN command:

```
COWBOY,N,V5,Y,Y,,Y,,,,,
```

Thus, the name of the user's object module is COWBOY; the source data set is not prestored; the first line number of the data set is the default value of 100; subsequent line numbers are incremented by the default value of 100; the version identification is V5 (not a time stamp); an internal symbol dictionary is to be produced, a source listing is to be produced (this parameter could have been defaulted with the same result); an object listing and cross-reference listing are not to be produced (the default options); a symbol table edit is to be produced; no storage map is to be produced (the default option); no BCD special characters are to appear in the input (the default option); and the CSECT of the module to be created is to have the private attribute (the default option).

Structure and Description of Compiler Listings

The compiler will prepare a listing if one or more of the five listing options are requested. The five types of listings are: source program listing, object program listing, cross-reference listing, symbol table listing, and storage map listing. The listings produced vary somewhat, depending upon the combination of listing options chosen.

The figures that follow show the listings produced for a compilation requesting the default listing options (source listing only), a compilation requesting all five listings and a compilation requesting only a storage map listing. A description is given for all sample listings. This description includes both a general discussion of items contained in a particular listing and references to particular items in the listing. Where appropriate, a reference number in bold-face type will be given in the listing description, for example, **1**. Such numbers refer to the encircled numbers on the listing itself that correspond to the item being discussed.


```

1 VERSION 07/10/67 OF THE TSS FORTRAN COMPILER ENTERED
2 THE MODULE NAME AND VERSION FOR THIS COMPIATION ARE COWBOY ,V5
3
4 OPTIONS--PUBLIC CSECT(N),BCD MODE(N),PRODUCE ISDIN). LISTINGS--SOURCE(Y),OBJECT(N),CROSS REF(N),SYMBOL TABLE(N),MEMORY MAP(N).
PAGE 001

```

Figure 9. Heading Page

```

5 COWBOY ,V5
6
7
100 C EXAMPLE PROGRAM TO DEMCNSTRATE TSS/360 FORTRAN COMPILER LISTINGS
200 DIMENSION A(10),B(10)
300 COMMON /NAME/C(5),D(5,2) /INDEX/L,M
400 DATA A/10*0.0/,B/10*1.0/
500 NAMELIST /NUM/L
600 5 DO 10 I=1,2
700 READ (5,NUM)
800 DO 10 J=1,L
900 READ (5,90C) D(I,J)
1000 10 CONTINUE
1100 CALL COMPUT(A,B)
1200 C DELIBERATE DIAGNOSTIC FOR ILLUSTRATIVE PURPOSES.
1300 X=A(11)
8 0001300 E *** ILLEGAL EXPRESSION IN SUBSCRIPT ON A
1400 IF(L.GE.M)GO TO 20
1500 WRITE (6,910) I,A(I),B(I),I=1,10)
1600 GO TO 5
1700 20 STOP
1800 900 FORMAT(2E20.8)
1900 910 FORMAT('1TEST PROGRAM SAMPLE OUTPUT FOLLEWS' /'O I
2000 X B*(16,E20.8,E20.8))
2100 END
PAGE 002

```

Figure 10. Source Program Listing

Heading Page

A heading page containing three lines is produced if any edits are requested. These heading lines describe: an identification of the version of the compiler being used **1**, the module name **2** and version (or time stamp, if version has been defaulted) for the compilation **3**, and a record of the options (other than verid) selected for the compilation **4**. An example of a heading page appears in Figure 9. In this example, the user supplied the module name COWBOY, version identification V5, and defaulted all other parameters.

Source Program Listing

If a source listing is requested, the first listing produced following the heading page is the source program listing. The source program listing will contain the diagnostic messages for each statement following the statement. A sample source program listing is shown in Figure 10. The first line of all listings contains the module name, **5**, and the version identification, **6**. All source lines are preceded by the system-assigned line number, **7**. A diagnostic message is included in the sample source listing **8**.

General Description of Output Module Listing

Following the source program listing, listings of the compiled object program module are given. In general, the listing of a TSS/360 FORTRAN output module follows the actual organization of the output module. For any

FORTRAN source program other than a BLOCK DATA subprogram, the compiler generates an output module that consists of at least two control sections — a CSECT¹ and a PSECT. The CSECT contains executable code and other information that will not be changed during program execution. The name of the CSECT is derived from the user-supplied module name by appending the two characters #C to the module name, or its leftmost six characters. The PSECT is a prototype control section that contains a register save area, address constants (adcons), parameter lists, NAMELIST information, non-COMMON variables, and local and global temporary storage. The name of the PSECT is generated in the same manner as the name of the CSECT, the suffix for the PSECT being #P. If COMMON areas are defined, each COMMON (blank and any named COMMON) is represented by a common control section in the object program module.

The BLOCK DATA subprogram generates an output module that contains one or more common control sections. There is no CSECT or PSECT.

Default Option Listing

If the default options are chosen, an abbreviated description of the CSECT and PSECT is given. The sample shown in Figure 11 contains header information and

¹The term "CSECT" generally refers to a control section other than a PSECT or a COMMON control section. In documentation or listings where compactness is necessary, the user may find CSECT referring to control sections in general.

ENTRY NAME	LOC HEX	EXTERNAL REFERENCES	CHCIA1	CHCIE1	CHCIU1	CHCIW2	COMPUT
COWBOY	00000000						
COWBOY#C				428 BYTES	07/21/67	21:44:12	
CODE					LOC HEX 00000000		356 BYTES
NUMERIC CONSTANTS					LOC HEX 00000170		60 BYTES
COWBOY#P				544 BYTES	07/21/67	21:44:12	
REGISTER SAVE AREA					LOC HEX 00000000		76 BYTES
CONVERSION CONSTANTS					LOC HEX 0000004C		24 BYTES
ADDRESS CONSTANTS					LOC HEX 00000064		108 BYTES
NAMLISTS					LOC HEX 00000138		28 BYTES
ALPHAMERIC					LOC HEX 0000015A		105 BYTES
NON-COMMON VARIABLES (TOTAL)					LOC HEX 000001C8		88 BYTES
NAME							60 BYTES
INDEX							8 BYTES

Figure 11. CSECT and PSECT Listings for Default Listing Options

LINE NO.	LABEL	LOC HEX	INST HEX	INST ASSEMBLER	COMMENTS
600	5	0C000034	5860D074	L 6,116(0,13)	NAME
700	599999	00000040	5810D084	L 1,132(0,13)	PARAMETER LIST
800		00000052	1886	LR 8,6	VARIABLE
900	599998	00000062	5810D090	L 1,144(0,13)	PARAMETER LIST

Figure 12. CSECT Listing

a summary of PSECT and CSECT information. In the header information, a line is given containing the module name **9** and the version identification, **10**. Next, the module name is repeated, **11** and the size in decimal of the module (CSECT plus PSECT) is given, **12**. The name of the entry point follows, **13** with the location of this entry point relative to the CSECT, **14**. Next are listed all external references made by the compiled program, **15**. Only one of the external references listed (COMPUT) is also found in the source program. The other external names all begin with the three letters CHC, which identifies them as FORTRAN iv-supplied subprograms called by the object program. A list of all such programs, with their names, is given at the end of this appendix.

The name of the CSECT, **16**, and the number of storage locations in the CSECT, **17**, are then given. Also contained on this line is the time stamp associated with the CSECT, **18**. This time stamp is the same as that supplied for version identification, if the user defaulted this compilation parameter, or is obtained by the compiler for identification of the CSECT. Following this line is the size, **19**, and relative location, **20**, of the two types of information contained in the CSECT.

The next section gives a summary of the PSECT contents. The line containing the name, **21**, and size, **22**, of the PSECT is followed by the relative location and size of the seven principal areas in the PSECT, **23**.

Detailed Description of Output Module Listing

If an object code listing is requested, a detailed listing of the CSECT and PSECT (or COMMON control section, for BLOCK DATA subprograms) is given. Figures 12 and 13 contain examples of CSECT and PSECT listings, respectively. In addition, a listing of initialized variables is given. Figure 13A contains an example of the Table of Initialized Variables. The three listings are described below.

Header Information: Each page of the listing is headed by a line containing the user-specified module name, the version, and the page number, **24**. The next line in the listing contains the module name, **25**, and the module size in decimal, **26**. This size is the total number of bytes occupied by the CSECT and the PSECT, excluding COMMON control sections. This information is followed by a table of all entry names, **27**, and their locations (hexadecimal) relative to the base of the CSECT, **28**. The header information is completed by a list of all references to external routines, **29**.

Description of CSECT Listing: A line containing the name, **30**, and size, **31**, of the CSECT is printed immediately preceding the code. Next is a line giving

the location of the executable code relative to the CSECT, **32**, followed by the size in decimal of the executable code, **33**. Following this line are six columns, described below.

1. *Line Number* **34**: The line numbers correspond to line numbers in the source data set. Each line number appears on the same print line as the first executable instruction of the respective source statement. Line numbers of nonexecutable statements (DIMENSION, COMMON, DATA, etc.) are omitted. However, CONTINUE statements are included in the listing.
2. *Label* **35**: Any entry name, statement number, or compiler-generated label is printed on the same line as the first instruction of the corresponding source statement that defines such a label. Compiler-generated labels indicate the destination of some compiler-generated branches. All compiler labels are six-digit numbers and are generated in a decreasing sequence starting with 999,999. (In some cases, the compiler-generated labels are not in descending sequence, and gaps may occur in the descending sequence due to compiler code optimization.)
3. *Hexadecimal Location* **36**: This column gives the location of the first byte of each instruction relative to the base of the CSECT.
4. *Instruction (Hexadecimal)* **37**: Each instruction is represented by its machine language (hexadecimal) configuration.
5. *Instruction (Assembly Type)* **38**: Each instruction is represented by its assembler language equivalent. All items in the operand field are decimal integers. No extended mnemonics are used except in conditional branches, where the operand field contains the label of the branch terminal.
6. *Comments* **39**: Whenever the operand (in storage) of an RX- or RS-type instruction can be meaningfully described, the description is printed in the comments column of the appropriate line.

A referenced constant is commented by its value, which is printed in conventional FORTRAN representation, and by its location (hexadecimal) relative to the base of the CSECT. Each part — real and imaginary — of a complex constant is commented. Certain fixed binary constants can be commented upon with their values and storage locations, although the instructions do not actually refer to the constants but obtain them by other means (e.g., an SLL instruction).

References to variables and elements of arrays contain the word “variable” followed by the variable or array name. No attempt is made to identify a reference to a specified subscripted array variable

LINE NO.	LABEL	LOC HEX	INST HEX	INST ASSEMBLER	COMMENTS	
		0000008A	58F0D094	L 15,148(0,13)	EXTERNAL	CHCIE1
		0000008E	0DEF	EASR 14,15		
		00000090	58E0D0A4	L 14,164(0,13)	EXTERNAL	CHCIU1
		00000094	50E0D048	ST 14,72(0,13)		
		00000098	58F0DCA0	L 15,160(0,13)	EXTERNAL	CHCIU1
		0000009C	0DEF	BASR 14,15		
1000	10	0000009E	4180B014	LA 8,20(0,8)		
		000000A2	1989	CR 8,9		
		000000A4	4770502E	BNE 999998		
		000000A8	41606004	LA 6,4(0,6)		
		000000AC	1967	CR 6,7		
		000000AE	4770500C	BNE 999999	PARAMETER LIST	000000F8
1100		000000B2	5810D0A8	L 1,168(0,13)		
		000000B6	58E0D080	L 14,176(0,13)		
		000000BA	50E0D048	ST 14,72(0,13)	EXTERNAL	COMPUT
		000000BE	58F0D0AC	L 15,172(0,13)		
		000000C2	0DEF	BASR 14,15		
1400		000000C4	58604000	L 6,0(0,4)	VARIABLE	L
		000000C8	59604004	C 6,4(0,4)	VARIABLE	M
		000000CC	4780511E	BNL 20	PARAMETER LIST	00000100
1500		000000D0	5810D084	L 1,180(0,13)	EXTERNAL	CHCIA1
		000000D4	58E0D08C	L 14,140(0,13)		
		000000D8	50E0D048	ST 14,72(0,13)	EXTERNAL	CHCIA1
		000000DC	58F0D088	L 15,136(0,13)		
		000000E0	0DEF	BASR 14,15	EXTERNAL	CHCIE1
		000000E2	58E0D098	L 14,152(0,13)		
		000000E6	50E0D048	ST 14,72(0,13)		
		000000EA	1973	LR 7,3		
		000000EC	41000001	LA 0,1(0,0)	VARIABLE	I
		000000F0	50003004	ST 0,4(0,3)	PARAMETER LIST	00000110
	599996	000000F4	5810D088	L 1,184(0,13)		
		000000F8	1800	SR 0,0	EXTERNAL	CHCIE1
		000000FA	58F0D094	L 15,148(0,13)		
		000000FE	0DEF	BASR 14,15	PARAMETER LIST	00000118
		00000100	5810D08C	L 1,188(0,13)		
		00000104	41007030	LA 0,48(0,7)		
		00000108	50001004	ST 0,4(0,1)		
		0000010C	1800	SR 0,0	EXTERNAL	CHCIE1
		0000010F	58F0D094	L 15,148(0,13)		
		00000112	0DEF	BASR 14,15	PARAMETER LIST	00000120
		00000114	5810D0C0	L 1,192(0,13)		
		00000118	41007008	LA 0,8(0,7)		
		0000011C	50001004	ST 0,4(0,1)		
		00000120	1800	SR 0,0	EXTERNAL	CHCIE1
		00000122	58F0D094	L 15,148(0,13)		
		00000126	0DEF	BASR 14,15		
		00000128	41707004	LA 7,4(0,7)	VARIABLE	I
		0000012C	58603004	L 6,4(0,3)		
		00000130	41606001	LA 6,1(0,6)	VARIABLE	I
		00000134	50A03004	ST 6,4(0,3)	10 AT	0000018C
		00000138	5960201C	C 6,28(0,2)		

LINE NO.	LABEL	LOC HEX	INST HEX	INST ASSEMBLER	COMMENTS	
		0000013C	47D050C0	BNH 999996	EXTERNAL	CHCIU1
		00000140	58E0D0A4	L 14,164(0,13)		
		00000144	50E0D048	ST 14,72(0,13)	EXTERNAL	CHCIU1
		00000148	58F0DCA0	L 15,160(0,13)		
		0000014C	0DEF	BASR 14,15		
1600		0000014E	47F05000	B 5		
1700	20	00000152	5810D0C4	L 1,196(0,13)		
		00000156	58E0D0CC	L 14,204(0,13)		
		0000015A	50E0D048	ST 14,72(0,13)	EXTERNAL	CHCIW2
		0000015E	58F0D0C8	L 15,200(0,13)		
		00000162	0DEF	BASR 14,15		

(40) NUMERIC CONSTANTS

(41) LOC HEX 00000170 SIZE

(42) 60 BYTES

(43) TYPE

- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4
- I*4

(44) LOC HEX
 00000170
 00000174
 00000178
 0000017C
 00000180
 00000184
 00000188
 0000018C
 00000190
 00000194
 00000198
 0000019C
 000001A0
 000001A4
 000001A8

(45) CONTENTS HEX

- 00000000
- 00000001
- 00000002
- 00000005
- 00000004
- 00000014
- 00000006
- 0000000A
- 00000008
- 00000080
- 00000020
- 000000C0
- 00000033
- 00000040
- 00000032

Figure 12. CSECT Listing, Continued

(element of an array). This information must be obtained by inspecting the D2, X2, and B2 fields of the instruction itself. As in the case of constants, references to complex variables are fully commented upon in both parts.

Subroutine calls are described by the word “external” followed by the name of the subroutine. In a number of subroutine calls there can be two such comments.

Arguments, or, specifically, adcons that contain the base register¹ for arguments at object time, are indicated by the word “argument” and the name of the argument. The actual arguments themselves are identified as “variable” followed by the argument name.

References to local or global temporary storage are printed as “local temp” and “global temp,” respectively. Again, the location of the temporary storage item must be obtained from the operand field of the instruction.

Address constants are described by the names of the storage classes the adcons cover and the pre-relocation values of such adcons (zero values are not printed). The following items can appear in the comments column in connection with adcon references:

CODE	Base register adcon for executable code
NUMERIC	Base register adcon for numerical constants
PARAMETER LIST	Adcon that contains the base register for the parameter list in a subroutine call
LOCAL TEMP STGE	Local temporary storage base register adcon
GLOBAL TEMP STGE	Global temporary storage base register adcon
NON-COMMON VAR.	Base register adcon for variables not declared in any COMMON
BLANK COMMON	Blank COMMON base register adcon
Name of a named COMMON	Base register adcon for that named COMMON

All numerical constants are listed at the end of the csect (after the listing of executable code), **40**. The relative location of the numerical constants in the csect is given, **41**, followed by the length in decimal of the numerical constants, **42**. Each constant is described by its type and length, **43**, location within the csect, **44**, and its internal representation, **45**. The types are:

- I – fixed binary (integer)
- R – floating binary (real)
- C – floating binary (complex)

¹The term “base register adcon” is used to refer to an address constant that is loaded into a base register.

The length is given in bytes, thus, for example:

- I*4 means an integer constant of 4-byte length.
- C*16 means a complex constant (floating binary) with a *total* length of 16 bytes – 8 bytes for the real part and 8 bytes for the imaginary part.

Description of PSECT Listing

The information contained in the page heading line, **46**, and psect title line, **47**, is similar to that contained in the corresponding lines of the csect listing. Listed next is the relative location within the psect and the length of the areas.

A sample psect listing is shown in Figure 13.

1. *Register Save Areas* **48**: This area is used to preserve register contents upon executing a call to a subroutine.
2. *Conversion Constants* **49**: This area contains the necessary masks and working storage needed by some in-line conversions (e.g., fixed to floating).
3. *Address Constants* **50**, **51**: Each adcon is described by:
 - a. The location of the adcon relative to the base of the psect.
 - b. The contents (value) of the adcon prior to relocation.
 - c. The control section referred to by the adcon or, in the case of arguments and external references, the item referred to by the adcon.
 - d. The “storage class referenced” describes what part of the specified control section is covered by the adcon or what argument or external name corresponds to the adcon.

Following the description of the above three areas is a listing of the individual items contained in the psect. The first item in this listing is the address constants, **51**. The address constants are followed by six other groups of items, described below.

1. *Parameter Lists* **52**: The listing of items in the parameter list section of the psect is in a format identical to that of the adcon listing. A number of items may have no entries under “Control Section Referenced” and “Storage Class Referenced.” This indicates that the contents of such parameters are not known at compile time and that the parameter is computed and stored in the indicated location upon execution of the program.
2. *NAMELISTs* **53**: This area contains the internal representation of NAMELIST information. Only the total size is printed.
3. *Alphamerics* **54**: Any alphameric constants, including contents of FORMAT statements, are listed by giving the starting byte of the character string followed by the string itself.

46 COWBOY ,V5

47 COWBOY*P SIZE 544 BYTES 07/21/67 21:44:12
 48 REGISTER SAVE AREA LOC HEX 00000900 SIZE 76 BYTES
 49 CONVERSION CONSTANTS LOC HEX 0000004C SIZE 24 BYTES
 50 ADDRESS CONSTANTS LOC HEX 00000064 SIZE 108 BYTES

LOC HEX	CONTENTS HEX	CONTROL SECTION REFERENCED	STORAGE CLASS REFERENCED
00000064	00000034	CSECT	CODE
00000068	000001C8	PSECT	NON-COMMON VAR.
0000006C	00000170	CSECT	NUMERIC
00000070	00000000	INDEX	INDEX
00000074	00000000	NAME	NAME
00000078	00000220	PSECT	LOCAL TEMP STGE
0000007C	00000000	EXTERNAL	CHCRD1
00000080	00000000	EXTERNAL	CHCRD1
00000084	00000000	PSECT	PARAMETER LIST
00000088	00000000	EXTERNAL	CHCTA1
0000008C	00000000	EXTERNAL	CHCTA1
00000090	00000000	PSECT	PARAMETER LIST
00000094	00000000	EXTERNAL	CHCTE1
00000098	00000000	EXTERNAL	CHCTE1
0000009C	00000000	PSECT	PARAMETER LIST
000000A0	00000000	EXTERNAL	CHCTU1
000000A4	00000000	EXTERNAL	CHCTU1
000000A8	00000000	PSECT	PARAMETER LIST
000000AC	00000000	EXTERNAL	COMPUT
000000B0	00000000	EXTERNAL	COMPUT
000000B4	00000100	PSECT	PARAMETER LIST
000000B8	00000110	PSECT	PARAMETER LIST
000000BC	00000118	PSECT	PARAMETER LIST
000000C0	00000120	PSECT	PARAMETER LIST
000000C4	0000012C	PSECT	PARAMETER LIST
000000C8	00000000	EXTERNAL	CHCTW2
000000CC	00000000	EXTERNAL	CHCTW2

52 PARAMETER LIST LOC HEX 00000000 SIZE 100 BYTES

LOC HEX	CONTENTS HEX	CONTROL SECTION REFERENCED	STORAGE CLASS REFERENCED
00000000	0000017C	CSECT	NUMERIC
00000004	00000197	CSECT	NUMERIC
00000008	00000198	CSECT	NUMERIC
0000000C	00000138	NUM	NAMLIST
00000010	0000017C	CSECT	NUMERIC
00000014	00000197	CSECT	NUMERIC
00000018	0000019F	CSECT	NUMERIC
0000001C	00000158	PSECT	ALPHAMERIC
00000020	000001A3	CSECT	NUMERIC
00000024	00000000		
00000028	000001F8	PSECT	NON-COMMON VAR.
0000002C	00000100	PSECT	NON-COMMON VAR.

COWBOY ,V5

LOC HEX	CONTENTS HEX	CONTROL SECTION REFERENCED	STORAGE CLASS REFERENCED
00000100	00000188	CSECT	NUMERIC
00000104	000001A7	CSECT	NUMERIC
00000108	0000019F	CSECT	NUMERIC
0000010C	00000160	PSECT	ALPHAMERIC
00000110	000001A8	CSECT	NUMERIC
00000114	000001CC	PSECT	NON-COMMON VAR.
00000118	000001A7	CSECT	NUMERIC
0000011C	00000000		
00000120	000001A3	CSECT	NUMERIC
00000124	00000000		
00000128	00000000		
0000012C	00000128	PSECT	PARAMETER LIST
00000130	00000000		

53 NAMELISTS LOC HEX 00000138 SIZE 28 BYTES
 54 ALPHAMERIC LOC HEX 00000158 SIZE 105 BYTES

LOC HEX	ALPHA (2E20..B)			
00000158			A	B'/(16,E20..8,E20..81)
00000160	('1TEST PROGRAM SAMPLE OUTPUT FOLLOWS'/*0 I			

55 NON-COMMON VARIABLES (TOTAL) LOC HEX 000001C8 SIZE 88 BYTES

LOC HEX	VARIABLE	TYPE
000001C8	J	I*4
000001CC	I	I*4
000001D0	B	R*4
000001F8	A	R*4

56 NAME SIZE 60 BYTES

LOC HEX	VARIABLE	TYPE
00000000	C	R*4
00000014	D	R*4

INDEX SIZE 8 BYTES

LOC HEX	VARIABLE	TYPE
00000000	L	I*4
00000004	M	I*4

Figure 13. PSECT Listing

56A COWBOY ,VS					PAGE 008
56B TABLE OF INITIALIZED VARIABLES					
56C SECT	56D RELATIVE LOCATION	56E HEXADECIMAL VALUE	56F CONVERTED VALUE	56G VARIABLE	
COWBOY#P00C100	41100000	:	+0.100000F+01	B(1)	
	:	:	:	:	
0001F4	41100000	:	+0.100000F+01	B(10)	
0001FB	00000000	:	+0.000000F+00	A(1)	
:	:	:	:	:	
:	:	:	:	:	
00021C	00000000	:	+0.000000F+00	A(10)	

Figure 14. Table of Initialized Variables

COWBOY ,VS					PAGE 008
SYMBOL TABLE SCRT					
57 SYMBOL	58 TYPE	59 CLASS	60 STORAGE CLASS	+	OFFSET
A	R*4	ARRAY VARIABLE	NON-COMMON		00000030
B	R*4	ARRAY VARIABLE	NON-COMMON		00000008
C	R*4	ARRAY VARIABLE	NAME		00000000
CHCBD1		EXTERNAL NAME			
CHCTA1		EXTERNAL NAME			
CHCIE1		EXTERNAL NAME			
CHCIU1		EXTERNAL NAME			
CHCIW2		EXTERNAL NAME			
COMPUT		EXTERNAL NAME			
D	R*4	ARRAY VARIABLE	NAME		00000014
I	I*4	SIMPLE VARIABLE	NON-COMMON		00000004
J	I*4	SIMPLE VARIABLE	NON-COMMON		00000000
L	I*4	SIMPLE VARIABLE	INDEX		00000000
M	I*4	SIMPLE VARIABLE	INDEX		00000004
NUM		NAMELIST	NAMELIST		00000000
5		SOURCE LABEL	CODE		00000034
10		SOURCE LABEL	CODE		0000000F
20		SOURCE LABEL	CODE		00000152
900		FORMAT LABEL	ALPHAMERIC		00000000
910		FORMAT LABEL	ALPHAMERIC		00000008
999996		COMPILER LABEL	CODE		000000F4
999998		COMPILER LABEL	CODE		00000062
999999		COMPILER LABEL	CODE		00000040

Figure 15. Symbol Table Listing

4. *Temporary Storage*: As in the case of NAMELIST, local and global temporary storage blocks are identified on the listing by their starting locations (relative to the PSECT's base) and their respective sizes. The example contains no temporary storage blocks.
5. *Non-COMMON Variables 55*: The listing of the PSECT is completed by information about non-COMMON variables. The starting location of the non-COMMON variable block and its total size are given. Each variable (or array) is described by its location with respect to the base of the PSECT, the name, and the type-length code. The type-length codes have the same meanings as described under constants.
6. *Common Control Sections Listing 53*: Any COMMON block — blank or named — is identified by its name and the total size in bytes. All variables contained in a COMMON block are listed in the same format as non-COMMON variables. The locations are relative to the base of the respective COMMON block.

Description of Table of Initialized Variables

The page heading line, 56A, is identical to the heading on the CSECT and PSECT listings. The next line in the listing contains the listing title, 56B, followed by the column headings. The information contained in the five columns is described below.

1. *SECT 56C*: This column contains the module name specified by #P if a main program. If a BLOCK DATA subprogram is compiled, the name of the COMMON block will print. If more than one page of listing is required, the name will be repeated on the first line of data on each page.
2. *Relative Location 56D*: This column gives the hexadecimal value of the displacement of the variable within the section.
3. *Hexadecimal Value 56E*: This column gives the hexadecimal value of the preset variable as it appears in the text.
4. *Converted Value 56F*: This column contains the preset value converted according to type of con-

stant. These types are: hexadecimal, integer, real, complex, logical, and literal.

5. **Variable 56G** : The variable name with subscript, if applicable, prints in this column. Variables are listed in order by displacement within section.

Columns 2-5 may also contain a "3-dot" notation, **56H**, if the repetition factor of a preset value exceeds 5. In this case, only the first and last values of the range print on the listing.

Description of Symbol Table Listing

The symbol table listing (symbol table sort) contains an alphanumerically sorted listing of symbols — names of variables, entry points, external references, labels,

etc. Each symbol is printed on a line and is described as follows.

The name of the symbol, **57**, is followed by its type-length code, **58**. The code has a meaning for the item represented by the given symbol. Entries under "class" further describe the item, and they are self-explanatory, **59**. "Storage class + offset," **60**, identifies the storage class to which a given symbol belongs and the location within such storage class. Under "storage class" there can be references to blank COMMON, code, alphanumeric, name of a named COMMON, etc. External names are not part of the compiled module; therefore, no information can be given for storage class + offset.

A sample Symbol Table Listing is shown in Figure 15.

COWBOY ,VS		PAGE 000			
(61) SYMBOL CROSS-REFERENCE LIST		(63) REFERENCES			
SYMBOL	DEFINITIONS (62)				
A		200	400	1100	1500
B		200	400	1100	1500
C		300			
CHCRD1		0			
CHCTA1		700	900	1500	
CHCIE1		900	1500		
CHCIU1		900	1500		
CHCIW2		1700			
COMPUT		1100			
COWBOY		0			
D	900	300			
I	600 1500	900	1500	1500	1500
INDEX		300			
J	800	900			
L		300	500	800	1400
M		300	1400		
NAME		300			
NUM		500	700		
COWBOY ,VS		PAGE 010			
(64) LABEL CROSS-REFERENCE LIST		(66) REFERENCES			
LABEL	DEF (65)				
00005	600	1600			
00010	1000	600	800		
00020	1700	1400			
00900	1800	900			
00910	1900	1500			

Figure 16. Cross-reference Listing

Description of Cross-reference Listing

The cross-reference listing is divided into two sections: a listing of names (symbols), **61**, followed by a listing of labels, **64**. Both sections list the corresponding items in alphamerically sorted sequence. Line numbers, as defined in the source listing, are printed under "definitions," **62** and **65**, and "references," **63** and **66**, for each item. Whenever an item is defined (e.g., label of a labeled statement or variable name on the left side of an assignment statement), the corresponding line number is printed in the "definitions" column. Any reference to an item causes an entry under "references." Multiple entries for any item are sorted in order of increasing line numbers. Multiple references to the same item in the same source line result in multiple printing of the same line number for that item. Compiler-generated labels are not included in the listing.

A sample cross-reference listing is shown in Figure 16.

Description of Storage Map Listing

If an object code listing or both an object code listing and a storage map listing are requested, the listing produced will be as described in the detailed description of the output module listing above. If a storage map listing is requested and no object listing is requested, the resultant listing will contain summary information for the module, **67**, CSECT, **68**, and for the PSECT, **69**. This information includes the CSECT and PSECT sizes, external definitions and references, the location relative to the CSECT of all FORTRAN statement numbers, the size in the CSECT of the numeric constants area, the relative location and the size of the principle PSECT areas, and the location in the PSECT of all variable names.

A sample storage map listing is shown in Figure 17.

COWBOY ,VS		PAGE 003					
67	COWBOY	SIZE	972 BYTES				
	ENTRY NAME	LOC HEX					
	COWBOY	00000000					
	EXTERNAL REFERENCES						
	CHCRD1	CHCIA1	CHCIE1	CHCIU1	CHCIW2	COMPUT	
68	COWBOY#C	SIZE	428 BYTES		07/21/67 21:44:12		
	CODE		LOC HEX	00000000	SIZE	356 BYTES	
	LINE NO.	LABEL	LOC HEX				
	600	5	00000034				
	700	999999	00000040				
	800		00000052				
	900	999998	00000062				
	1000	10	0000009E				
	1100		000000B2				
	1400		000000C4				
	1500		000000D0				
		999996	000000F4				
	1600		0000014E				
	1700	20	00000152				
	NUMERIC CONSTANTS		LOC HEX	00000170	SIZE	60 BYTES	
COWBOY ,VS		PAGE 004					
69	COWBOY#P	SIZE	544 BYTES		07/21/67 21:44:12		
	REGISTER SAVE AREA		LCC HEX	00000000	SIZE	76 BYTES	
	CONVERSION CONSTANTS		LCC HEX	00000040	SIZE	24 BYTES	
	ADDRESS CONSTANTS		LCC HEX	00000064	SIZE	108 BYTES	
	NAMELISTS		LCC HEX	00000198	SIZE	28 BYTES	
	ALPHAMERICS		LCC HEX	00000158	SIZE	105 BYTES	
	NON-COMMON VARIABLES (TOTAL)		LOC HEX	000001C8	SIZE	88 BYTES	
	LOC HEX	VARIABLE	TYPE				
	000001C8	J	I*4				
	000001CC	I	I*4				
	000001D0	R	R*4				
	000001F8	A	R*4				
	NAME		SIZE	60 BYTES			
	LOC HEX	VARIABLE	TYPE				
	00000000	C	R*4				
	00000014	D	R*4				
	INDEX		SIZE	8 BYTES			
	LOC HEX	VARIABLE	TYPE				
	00000000	L	I*4				
	00000004	M	I*4				

Figure 17. Storage Map Listing

Compilation Completed Message: Following production of requested listings, the page is restored and the message `COMPILATION COMPLETED` is written.

Destination of Compiler-Produced Listings

The destinations of compiler-produced listings depend on whether the task is conversational or nonconversational, and on the value of the `FTN` command's `LISTDS` operand.

Conversational Tasks

In conversational tasks, all compiler-produced listings are placed in the list data set unless the `LISTDS` operand of the `FTN` command specifies `SYSOOT`.

Printing of the list data sets prepared by the compiler is not automatic. Each time a unique module name is encountered, a generation data group is established, containing two generations. Each time the limit (two generations) is reached, the oldest generation is erased. The user may print only when he wants the output listings, using: `PRINT LIST.module-name` followed by the relative or absolute generation. The *Command System User's Guide* presents a complete explanation of the language processor listing data set maintenance process.

Since a pending `BULK/IO` task will be established when the `PRINT` command is issued for the language processor listing data set, the user must not attempt to erase the data set (or otherwise remove it from the system) unless the `BSN` is canceled first.

Nonconversational Tasks

In nonconversational tasks, the system automatically puts compiler-produced listings on `SYSOOT` and prints them. After printing, the listing no longer exists in the system.

This system action can be overridden, however, by the `LISTDS` operand of the `FTN` command. If you specify `LISTDS=Y`, the system puts the listing in the list data set and maintains it exactly as in a conversational task.

You can have the listing put in the list data set and printed immediately by specifying `LISTDS=Y` and following the compiler source statements with an appropriate `PRINT` command.

Note that if you use a `PRINT` command in a nonconversational task initiated from the terminal, you can always cancel the printout by issuing a `CANCEL` command. However, the `CANCEL` command does not prevent printing of a listing, or any part of a listing, that is already waiting in the `SYSOOT` data set.

FORTRAN IV Library Subprograms: Indirect References

Most FORTRAN IV library subprograms are referred to in a compiled program directly — i.e., by the same name used in the source program; the statement

$$X = \text{SIN}(Y)$$

for example, leads to a call on the `SIN` program, and `SIN` will be listed under external references in an object code listing. Certain other FORTRAN library subprograms are referred to by names created by the compiler. These programs are listed as follows, with their names.

<code>CHCBGA</code>	Raises an I°4	number to an I°4 power
<code>CHCBGB</code>	Raises an I°2	number to an I°2 power
<code>CHCBGC</code>	Raises an I°2	number to an I°4 power
<code>CHCBGD</code>	Raises an I°4	number to an I°2 power
<code>CHCBHA</code>	Raises an R°4	number to an I°4 power
<code>CHCBHB</code>	Raises an R°4	number to an I°2 power
<code>CHCBIA</code>	Raises an R°8	number to an I°4 power
<code>CHCBIB</code>	Raises an R°8	number to an I°2 power
<code>CHCBJA</code>	Raises an R°4	number to an R°4 power
<code>CHCBJB</code>	Raises an I°2	number to an R°4 power
<code>CHCBJC</code>	Raises an I°4	number to an R°4 power
<code>CHCBKA</code>	Raises an R°8	number to an R°8 power
<code>CHCBKB</code>	Raises an I°2	number to an R°8 power
<code>CHCBKC</code>	Raises an I°4	number to an R°8 power
<code>CHCBKD</code>	Raises an R°4	number to an R°8 power
<code>CHCBKE</code>	Raises an R°8	number to an R°4 power
<code>CHCBMA</code>	Raises a C°16	number to an I°4 power
<code>CHCBMB</code>	Raises a C°16	number to an I°2 power
<code>CHCBCA</code>	Raises a C°8	number to an I°4 power
<code>CHCBCB</code>	Raises a C°8	number to an I°2 power
<code>CHCBD1</code>	Initialize interrupt processing.	
<code>CHCIA1</code>	Initialize for an I/O call.	
<code>CHCIE1</code>	Transmit an I/O value.	
<code>CHCIU1</code>	Terminate an I/O call.	
<code>CHCIV1</code>	DUMP program	
<code>CHCIV2</code>	PDUMP program	
<code>CHCIW1</code>	EXIT program	
<code>CHCIW2</code>	STOP program call.	
<code>CHCIW3</code>	PAUSE program call.	
<code>CHCIW5</code>	Called if FORTRAN subprograms are erroneously entered at their standard entry point.	

Reference To Subroutines

Special considerations must be made when a FORTRAN main program makes references to subprograms. If a main program and its associated subprograms are compiled together, and an error is detected in one of the subprograms, that subprogram must be recompiled. The recompiled output module may be placed in the same program library as the original or in a different one. Depending on which method is chosen, the results will vary.

If the recompiled module is placed in the same program library, the user is asked if the module is a replacement. If it is, (the default condition) the system tries to unload any module in the user's virtual

storage that has the same name. The unloader will find a module but will not be able to unload it because of outstanding references. Unless the user explicitly issues an UNLOAD command for his main program, he will not get his new copy of the sub-program.

If the user defines a new program library, he will not be asked if the module is a replacement. Consequently, if he reruns the program, he will use the old copy since it is still loaded.

The simplest solution is for the user to unload the main program before he starts recompilation. He can also unload modules that have references to the new module, but this might not work since the modules he tries to unload might satisfy references in other modules. If the user unloads his main program, he avoids getting diagnostics from the loader during compilation and ensures that the latest level of the module is being used.

Destination of Output

Table 3 shows the destination of all output from any compilation variation.

Compiler Restrictions

Limitations of virtual storage available to the compiler and the object programs generated by it impose a number of restrictions on the size of a source program capable of being compiled. These restrictions are categorized according to complexity. The first category, simple source program restrictions, can easily be applied to individual source statements or particular types of source statements. Simple source program restrictions are listed in Table 4.

The term "file" is used in this section to refer to compiler work areas.

The second category, complex restrictions, is composed of restrictions that generally are too complex to anticipate in advance of compilation (e.g., the storage requirements of the various tables internal to the compiler are, in many cases, extremely difficult to compute accurately, as the table sizes are complex functions of the source program). Very few programs are of such a size or configuration that these complex restrictions can be met. Therefore, the FORTRAN user may not wish to concern himself with the complex restrictions until he receives a diagnostic message; then he can proceed to remedy the situation. Complex source program restrictions are listed in Table 5.

Should the user wish to more accurately determine the number of entries that can be made in the files listed in Table 5, the following paragraphs provide information allowing him to do so. For each file listed

in Table 5, either a formula, an absolute number, or an average number is given as a measure of the number of storage locations (bytes) in the file.

	<i>Storage Locations</i>
<i>Symbol Table</i>	
Variable or function-name	28
Four-byte constant	16
Eight-byte constant	20
Sixteen-byte constant	28
Label	20
Adcon	16
<i>Storage Locations</i>	
<i>Program Representation File (PRF)</i>	
Equation	16
Unconditional GO TO	16
IF	28
<i>Locations</i>	
<i>Program Representation File (PRF) con't.</i>	
CALL	20
DO	64
Average I/O Statement (Five list elements, two of them subscripted)	74
Average Label Definition	12
<i>Storage Locations</i>	
<i>Expression File (EF)</i>	
Average Equation (Ten variables and operators)	80
Unconditional GO TO	0
Average IF (Arithmetic IF)	80
Average CALL (One simple argument)	80
Average DO	0
Average I/O Statement (Five list elements, two of them subscripted)	100
Average Label Definition	0
<i>Storage Locations</i>	
<i>Storage Specification List (SPL)</i>	
Average COMMON (Three variables, arrays, or block names)	20
Average EQUIVALENCE (Three variables)	30
<i>Cross Reference Table (CRL)</i>	
CRL size = (number of occurrences of statement numbers + number of occurrences of names) x 8	
<i>Storage Locations</i>	
<i>Preset Data Table (PDT)</i>	
The PDT size is estimated from the relation: PDT size = CRL size (described above) + NAMELIST size (described below) + total for the following four types of statements.	
Average DIMENSION (Two arrays of three dimensions each)	24

Table 5. Complex Source Program Restrictions

Item (Storage Limit ¹)	Overflow Cause	Comments	Associated Diagnostic Message(s)	Corrective Action ²
Symbol Table (limited to 20 pages; 81,920 bytes)	Too many names, constants, and statement numbers in the source program.		SYMBOL TABLE OVERFLOW. SPLIT PROGRAM AND COMPILE PARTS SEPARATELY.	
Program Representation File and Expression File (limited to 60 pages; 245,760 bytes)	Source program too large.	Excessive use of nested statement function references and multidimensional subscripted variables (whose subscripts cannot be resolved at compile time) may contribute significantly to the size of this area.	SYMBOL TABLE OVERFLOW. TOO MANY NAMES, CONSTANTS, AND FORMATS. SYMBOL TABLE OVERFLOW IN PHASE 2. RETURN IS TO hex location	
Storage Specification List (limited to 60 pages; 245,760 bytes)	Too much COMMON or EQUIVALENCE information.		FILE OVERFLOW. TOO MANY EXECUTABLE STATEMENTS	
Cross Reference Table (see Present Data Table) (limited to 32 pages; 131,072 bytes)	Defining and referencing too many variables and statement numbers.	In programs containing a large number of entries in DATA statements, Preset Data table storage requirements may be reduced by use of repeat constants whenever possible.	TABLE OVERFLOW. TOO MUCH COMMON AND EQUIVALENCE INFORMATION CROSS-REFERENCE TABLE OVERFLOW	Do not request a Cross Reference Listing.
Statement Function Expression File (limited to 110,480 bytes)	Too many initial values, name lists, and formats.		TABLE OVERFLOW. TOO MANY INITIAL VALUES, NAME-LISTS, AND FORMATS	Use Block Data subprograms to preset initial values.
Non-COMMON Variable Storage	The cumulative size of statement function definitions is too large.		TABLE OVERFLOW. TOO MANY STATEMENT FUNCTIONS	Use function subprograms to define the larger functions.
COMMON Variable Storage	Too many non-COMMON variables in the source program.		STORAGE ASSIGNMENT TO NON-COMMON VARIABLE NAME IS TOO LARGE	
Program File (limited to 60 pages; 245,760 bytes)	Too many COMMON variables in the source program.		PROGRAM FILE OVERFLOW IN PHASE 3	
Code File (limited to 60 pages; 245,760 bytes)	Source program too large.	Excessive use of deeply nested statement functions, multidimensional subscripted variables, or complex arithmetic may contribute significantly to the size of this area.	CODE FILE OVERFLOW. SPLIT PROGRAM AND COMPILE PARTS SEPARATELY.	
Adcon Page (49% bytes)	Source program too large.	Total of all statement numbers in assigned GO TO statements, the number of input/output lists, and the number of function references contribute significantly to the size of this area.	ADCON PAGE OVERFLOW. SPLIT PROGRAM AND COMPILE PARTS SEPARATELY.	
Phase 2 Internal Table	Too many non-COMMON variable names and/or too many names in EQUIVALENCE statements.	The available space for sorting non-COMMON variables in determining storage assignments, or for solving EQUIVALENCE relationships, has been exceeded.	INTERNAL TABLE OVERFLOW IN PHASE 2. RETURN IS TO hex location	
Optimization Table (limited to 16,384 bytes)	Source program too large.	Contributors to this table are primarily retained in the size of the source program, particularly the number of DO loops and the extent of subscripted variable storage.	WORKING STORAGE OVERFLOW IN PHASE 3.	
Program Module Dictionary (limited to 6 pages; 24,372 bytes)	Source program too large.	The amount of storage specified in DIMENSION statements, the number of subprogram references, and the number of I/O statements contribute significantly to the size of this area.	PROGRAM MODULE DICTIONARY TABLE OVERFLOW	
Name Table (External Name List) (limited to 2 pages; 8,192 bytes)	Too many ENTRY statements.		NO. OF ENTRY OR COMMON NAMES HAVE CAUSED A NAME TABLE OVERFLOW	Reduce the number of ENTRY statements or named COMMON names.
Internal Symbol Dictionary (limited to 10 pages; 40,960 bytes)	Too many variables and source statements.		INTERNAL SYMBOL DICTIONARY OVERFLOW	Do not request an Internal Symbol Dictionary.
Object Program Module (limited to 42 pages; 172,032 bytes)	Source program too large.		OBJECT PROGRAM MODULE EXCEEDS ALLOCATED SIZE	
Name List (common and removed expression table)	There may be only 4095 distinct expressions (including sub-expressions) resolvable as common or removable from loops. The maximum number has been exceeded.		NAME LIST DELETED IN PHASE 3.	
Number of items in the internal linear form of a subscript is limited to 164 divided by (the number of dimensions + 1)	The maximum number of arguments in a function or subroutine subprogram is limited to 126. ³		TABLE OVERFLOW. SUBSCRIPT EXPRESSION IS TOO BIG	Use an action menu statement to compute all or part of the subscript prior to its use as a subscript. Do not request a Cross Reference Listing.
Number of primitives (function names, constants, and variable arguments to function references) that appear in a subscript and 2500. Generally there is a one-to-one correspondence between the elements of an expression in the source statement and the internal representation. There are two cases where this is not true, and where the internal representation may become quite large:			FORMAL ARGUMENT ADCON TABLE OVERFLOW IN PHASE 3.	
1. Statement function references.	1. Statement function references are expanded inline; therefore, the use of deeply nested statement functions may contribute significantly to the size of the internal representation of the expression.		EXPRESSION TOO BIG	Divide the statement into several smaller statements.
2. Subscripts.	2. Variable subscripts, and subscripts with variable dimensions, are expanded to a linear form to be compared during object program execution. The use of multidimensional subscripted variables may contribute significantly to the size of the internal representation of the expression.			

¹ Where an explicit limit is not given, the limit is a complex function of other limits.

² General corrective action is to divide the program into a main program and one or more subprograms, and compile the parts separately.

³ Other corrective actions are noted in the "Corrective Action" column, and may in some cases be implied from information in previous columns.

This restriction is listed here as well as in Table 4 as in certain cases the restriction may be a complex one.

Table 3. Destination of Compiler Output

COMPILER VARIATION	OUTPUT			
	OBJECT MODULE	SOURCE	LISTINGS	COMPILER DIAGNOSTICS
Conversational — Input from terminal keyboard or card reader.	Latest job library defined in task, or USERLIB.	Data set named SOURCE. module-name, created by system.	To list data set member named LIST.module. To SYSOUT when user issues appropriate PRINT command.	To terminal, and to list data set if listings requested.
Conversational — Prestored data set.		Data set named SOURCE. module-name, will be updated to reflect modifications.		
Nonconversational — Prestored data set.			To list data set if requested; otherwise to SYSOUT.	To SYSOUT data set if no listings requested; otherwise, to list data set only.
Nonconversational — Input after FTN.		Data set named SOURCE. module-name, created by system.		

Average COMMON (Three variables, arrays, or block names) 24
 Average Explicit Type Statement (Three variables, arrays, or function names, each with one initial data value) 72
 Average DATA Statement (Four variables, each with one value) 64

Statement Function Expression File Storage Locations
 Average equation (Ten variables and operators) 80

Non-COMMON Variable Storage

The storage assignment for any variable may not exceed $2^{24}-1$ relative to the assignment of the first non-COMMON variable assigned. The assignment of this variable is independent of its own length (save for at most 15 bytes of padding which might be necessary for proper internal alignment), but depends upon the assignments made by the compiler to earlier processed variables and upon the additional effect of an EQUIVALENCE statement which might involve this variable.

COMMON Variable Storage

The storage assignment for any variable may not exceed $2^{24}-1$ relative to the origin of the COMMON block within which it is to be assigned. If this variable appears in a COMMON statement, then the sum of the lengths of the variables and arrays that were declared previously in the same COMMON block is 2^{24} or greater. If the variable has been given EQUIVALENCE to a COMMON variable, then the EQUIVALENCE relationship is such as to make the assignment of the variable exceed $2^{24}-1$.

Table 4. Simple Source Program Restrictions

ITEM	MAXIMUM NUMBER OR SIZE	ASSOCIATED DIAGNOSTIC MESSAGE
Number of arguments in a statement function	25	ILLEGAL NUMBER OF ARGUMENTS USED FOR FUNCTION name
Number of nested statement function references	24	STATEMENT FUNCTION CALLS NESTED TOO DEEP
Number of levels of nested DO loops (both explicit and implicit)	55	DO LOOPS NESTED TOO DEEP
Number of statement numbers in a computed or assigned GO TO	255	NUMBER OF STATEMENT NUMBERS EXCEEDS 255
Number of named COMMON blocks	118	TOO MANY COMMON BLOCKS
Number of significant characters ¹ in a source statement	1320	MORE THAN 1320 CHARACTERS THIS AND FOLLOWING CONTINUATION LINES WILL BE IGNORED
Total size of an array	$2^{24}-1$ bytes	ARRAY name IS TOO BIG
Number of characters in a character string	255	CHARACTER STRING TOO LONG
Maximum number of arguments in a sub-program reference ²	126	FORMAL ARGUMENT ADCON TABLE OVERFLOWED IN PHASE 3
Maximum number of dimensions in an array	7	variable HAS MORE THAN SEVEN DIMENSIONS
Maximum number of preset variables	510	WORK AREA OVERFLOW — INCOMPLETE TABLE OF INITIALIZED VARIABLES

¹See "Format of Source Lines" for the rules to determine the number of significant characters in a source statement.
²See also the more detailed discussion of this restriction in Table 5.

Program File (PF)

Estimated from the relation:

$$\text{PF size} = (\text{PRF size} + \text{EF size}) \times 0.9$$

Code File (CF)

Estimated from the relation:

$$\text{CF size} = \text{PF size} \times 0.5$$

Adcon Page

The adcon page contains certain address constants required by the compiled program. The particular address constants included in this page are implied by the items listed below as contributing to space required in the page.

The size of the adcon page can be estimated from the relation:

- (number of pages of constants referred to
- + number of pages of common or non-common variables or arrays referred to
- + number of pages of generated code estimated and referred to
- + number of pages of local temporary storage referred to
- + number of pages of global temporary storage referred to
- + number of literal occurrences of statement numbers in assign statements
- + number of formal parameters
- + number of pages relative to a formal parameter referred to except the first page
- + number of literal occurrences of external function references or external subroutines referred to with at least one actual argument
- + number of list elements) $\times 4$
- + (number of distinct external subprograms referenced) $\times 8$
- + 8 if there are any i/o statements
- + 16 if any i/o statement has a list

Phase 2 Internal Table

The following relation gives an upper bound on the size of entries in the table. Areas in the table are re-used frequently.

Internal Table size = (number of occurrences of variables excluding common, equivalence, and formal arguments

- + number of variables in equivalence statements
- + number of groups of variables in equivalence statements
- + number of occurrences of a variable common between groups of variables in equivalence statements) $\times 8$

+ (number of DO statements - 1 + number of branches into and out of DO loops) $\times 4$

Optimization Table (Triad Table)

If no ISD requested, estimated size of optimization table = number of executable statements $\times 36$

If ISD requested, estimated size of optimization table = number of executable statements $\times 72$

Program Module Dictionary

Estimated from the relation:

$$\begin{aligned} \text{PMD size} &= (\text{number of COMMON statements} + 2) \\ &\times 84 \\ &+ (\text{number of ENTRY statements}) \times 28 \\ &+ (\text{number of I/O statements}) \times 44 \\ &+ (\text{number of CALL statements} \\ &+ (\text{number of CALL statements} + \text{number of} \\ &\text{external subprogram arguments which are any} \\ &\text{of: variable or array name that is not a formal} \\ &\text{parameter; expression; constant; function name} \\ &\text{that is not a formal parameter}) \times 8 + 88 \end{aligned}$$

Name Table or External Name List (ENL)

ENL size = (number of COMMON statements + number of ENTRY statements + 3) $\times 8 + 4$

Internal Symbol Dictionary (ISD)

Estimated from the relation:

$$\begin{aligned} \text{ISD size} &= (\text{number of COMMON statements} + 2) \\ &\times 16 \\ &+ (\text{number of executable statements}) \times 8 \\ &+ (\text{number of variables} + \text{number of COMMON} \\ &\text{statements} + 2 + \text{number of FORMAT statement} \\ &\text{numbers}) \times 24^1 + 24 \end{aligned}$$

Object Program Module (Text)

Estimated from the relation:

$$\text{OPM size} = (\text{number of statements}) \times 24$$

NAMELIST

NAMELIST size = (number of names in NAMELIST statements) $\times 2$

The restrictions listed in Tables 4 and 5 are implementation restrictions; they apply only to this particular implementation of the FORTRAN language. They are a supplement to the language restrictions given in the publication *IBM FORTRAN IV*. Other than restrictions specifically stated in Tables 4 and 5, the only implementation restrictions placed upon the language are the limit of source statement size and maximum number of dimensions in an array.

¹Assuming average of one dimension per variable.

next command is read from `sysin`). If the statement contains an `AT` command (termed a dynamic statement), the actions are performed when the `FORTTRAN` statement number given in the `PCS` statement is reached during program execution. He can then enter any commands he wishes, including dynamic statements that are to be effective during execution. The `CALL` command, when used without an operand, executes the last module referred to by the system. If an object module is loaded after the main program is loaded, the name of the main program should be specified as the operand of the `CALL`.

During execution, statements can be entered whenever execution is interrupted, which can be the result of a `PAUSE` statement or of the entry, prior to the `CALL`, of a dynamic `PCS` statement containing a `STOP` command. The conversational user can also interrupt the execution of a program by pressing the attention button. The `STOP` command can then be entered to obtain the symbolic location in the program that is to be executed when the `GO` command is used to resume execution. (Refer to Appendix F for considerations in the use of the attention button.)

Checkout operations can be continued following execution until any module referred to by a dynamic statement is unloaded, at which time all `PCS` statements are removed from the user's object program modules.

Associated with each dynamic `PCS` statement is a counter that is incremented by one for each occurrence of the events specified in the statement. This counter can be referred to by the special character `%`. The value of the counter can be displayed or dumped and can be used in forming expressions. The counter (`%`) referred to is always the one associated with the statement in which it is referenced.

Since `%` is not a user's variable, it cannot be changed by a `SET` command.

Conversational Mode

`PCS` commands entered at the user's terminal (`sysin`) are immediately checked for valid syntax. References to variable names and statement numbers are checked in the appropriate module's `ISD`. Syntax errors and references to undefined symbols are reported to the user via diagnostic messages.

All `PCS` output is printed at the user's terminal (`sysout`), except for the `DUMP` command output, which is written on the `PCSOUT` data set.

Nonconversational Mode

`PCS` can be used in nonconversational mode with the following differences:

1. Erroneous `PCS` commands produce a diagnostic on the task's `sysout` data set, and the commands are ignored.
2. No prompting is performed, and incorrectly entered commands are ignored.
3. `PCS` output goes to the task's `sysout`; it is interspersed with user and system responses. `DUMP` command output is written on the `PCSOUT` data set.
4. If object program execution is interrupted by a `STOP`, the next command is taken from the task's `sysin`.

Notation

`PCS` commands consist of directives, operators, symbols, and constants. In a `PCS` statement, these elements are delimited by blanks. That is, blanks cannot be embedded in variables or constants, but they can be used following a comma, semi-colon, and around arithmetic, relational and logical operators, and parentheses used for grouping.

The character set is:

1. The letters A-Z (upper or lower case) and \$ # @.
2. The digits 0-9.
3. The special characters + - , > < \neg | = . * () ; ' / & % : blank.

Directives

The `PCS` directives are `AT`, `DISPLAY`, `DUMP`, `IF`, `QUALIFY`, `REMOVE`, `BRANCH`, `CALL`, `GO` and `STOP`. Each directive designates a `PCS` command.

Operators

Operators used to form arithmetic and logical expressions are:

TYPE	OPERATOR	MEANING
Arithmetic	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
Logical	\neg	Logical inversion or negation (NOT)
	&	Logical intersection (AND)
		Logical union (OR)
Relational	>	Greater than (GT)
	<	Less than (LT)
	=	Equal to (EQ)
	>=	Greater than or equal to (GE)
	<=	Less than or equal to (LE)
	\neq	Not equal to (NE)
	$\not>$	Not greater than (LE)
	$\not<$	Not less than (GE)

Symbols

Symbols refer to variable names, array names, and statement numbers. They are referred to in PCS with the same names and numbers used in the FORTRAN module.

PCS recognizes two kinds of symbols: external and internal.

FORTRAN external symbols are:

- Module name
- CSECT name (the module name suffixed with #C)
- PSECT name (the module name suffixed with #P)
- COMMON block names
- FUNCTION subprogram names
- SUBROUTINE subprogram names
- ENTRY names in subprograms

Internal symbols are those defined within a single compilation. FORTRAN statement numbers and FORTRAN variable or array names are internal symbols. Internal symbols can be referred to only if an ISD was requested when the module was compiled. Further, each internal symbol must be qualified to specify the program name in which the symbol was defined.

Certain names that appear in a FORTRAN source program are not available for use in PCS statements. These are names whose *only* occurrences in the source program are as any of the following:

1. As a formal argument in a SUBROUTINE, FUNCTION, or ENTRY statement.
2. As a dummy argument in a statement function.
3. As a type statement component without dimensions or initial values.

Names used only in these ways are ignored by the compiler. No storage is allocated for them, and no ISD entries are made.

Symbols can be qualified explicitly or implicitly. The internal symbol A is qualified explicitly as follows:

```
MAIN.A
```

An internal symbol is qualified implicitly as follows:

```
QUALIFY MAIN
```

```
.
```

```
.
```

```
.
```

```
DISPLAY A
```

If a module has been link-edited, internal symbols can only be used if an ISD has been requested as a linkage editor option and in addition, an ISD was requested at compile time for the module defining the internal symbol. Each internal symbol in a link-edited module, when referred to in a PCS statement, must be qualified by both the module name originally as-

signed at compilation time, and the name assigned to the link-edited output module. For example, if object module MAIN was link-edited into an output module name TOTAL, the internal symbol A in MAIN is explicitly qualified as:

```
DISPLAY TOTAL.MAIN.A
```

To use implicitly qualified internal symbols, both object module names must be specified in the QUALIFY:

```
QUALIFY TOTAL.MAIN  
DISPLAY A
```

The QUALIFY remains in effect until another QUALIFY is given. Explicitly qualified symbols can still be entered at any point.

The special symbol %COM can be used to refer to blank common; %COM can be used as either an external or internal symbol.

PCS commands can refer to dummy arguments to subprograms. The values of the arguments used are those established at the most recent execution of the subprogram when the PCS action is performed. Dummy arguments should not be referred to if the action requested in the command is to be performed prior to entry to the subprogram.

FORTRAN Statement Numbers

FORTRAN statement numbers are those written by the user in the original source program and should not be confused with the line numbers that are assigned to each source line by the compiler. Statements must be referred to by their statement numbers, not line numbers. Executable statement numbers used as internal symbols can be incremented to refer to unnumbered statements. The increment must be an integer greater than zero, enclosed in parentheses, and immediately following the statement number. The increment (1) refers to the numbered statement itself. Therefore, S6(1) refers to numbered statement S6; S6(2) refers to the next *executable* statement following numbered statement S6.

Executable statements are arithmetic and logical assignment statements, control statements, and input/output statements. Nonexecutable statements are specification statements and subprogram statements. Non-executable statements should not be incremented. For example, in the following statements:

```
10 READ (1,20)A  
20 FORMAT (F6.2)  
   B=A*3.14  
   WRITE (2,20)A,B  
   GO TO 10
```

to refer to the third statement ($B = A * 3.14$), .10(2) must be used; the FORMAT statement cannot be incremented since it is not executable.

Statement numbers refer to a statement's first line, plus any continuation lines; therefore, continuation lines need not be considered when using incremented statement numbers.

If the first executable statement in a program is unnumbered, the integer zero can be used to refer to it. In the above example, if the READ statement were unnumbered, 0 could be used to refer to it; 0(2) would then be used to refer to the second executable statement (i.e., $B = A * 3.14$).

Subscripted Symbols

Internal symbols that refer to arrays can be subscripted to refer to elements of their arrays. A subscript can be any integer arithmetic expression that does not itself include a subscripted symbol. The subscript is enclosed in parentheses, following the internal symbol naming the array. One subscript expression must be used for each dimension; multiple subscripts are separated by commas.

An array that is a dummy argument to a subprogram can be subscripted. The dimensions of the array are as defined in the subprogram. When an array has adjustable dimensions, both the array and the dimension values used are those established at the most recent execution of the subprogram.

Constants

Five classes of constants can be used to form expressions in PCS commands: integer, real, hexadecimal, character, and address constants. There are no complex constants.

1. Integer constants are expressed in the same manner as FORTRAN constants. For example: 9327, -642, +1066, -67.
2. Real constants are expressed in the same manner as FORTRAN constants. For example: 5764.1, 7.0E3, 16.9D-03, +0. .
3. Hexadecimal constants are written with one or more hexadecimal digits (0-9 and A-F), preceded by an X, and enclosed by apostrophes. For example: X'76543210', X'FFFFFFFF', X'AC7', X'9FEC3', X'00FF'. A hexadecimal constant is either truncated on the left or filled with zeros on the left if its length is inappropriate for the expression in which it appears.
4. Character constants can contain all letters, decimal digits, and special characters. Those remaining unused combinations (of the 256 card punch combinations) that can be designated in a character constant entered on cards are described in *Terminal User's Guide*. A character constant is enclosed in apostrophes. If an apostrophe is desired

as a character in a character constant, it must be represented in the written statement by a pair of apostrophes, although only one apostrophe will appear in storage. For example: '83.98', 'HOW ARE YOU?', 'I 'M FINE'. If the length of the constants is not appropriate for the context in which it occurs, the constant is truncated or filled with blanks on the right.

5. Address constants consist of the character A followed by a symbol enclosed in apostrophes. The allowable symbols are: external symbol with or without offset, internal symbol with or without offset, and subscripted variable. The length of an address constant is always four bytes; its value is the address assigned to the symbol. Address constants are evaluated at the time they are used. The current value of any variable referenced as a subscript is used in computing the value of the address constant. As a result, the value of an address constant that contains a subscripted symbol may vary during program execution. For example: A'NAME', A'ARRAY(I,J)'.

Expressions

Arithmetic Expressions

Arithmetic expressions in PCS are similar in most respects to FORTRAN expressions. They can be used as subscripts, as value to which variables are to be set, or as values to be compared in logical expressions.

The special character (%) can be used in an arithmetic expression to refer to a counter incremented by arrivals at the control point(s) specified in an AT command (or a PCS statement that includes one or more AT commands).

There is no exponentiation or function evaluation in PCS expressions.

The rules for formation of an arithmetic expression are as follows:

1. Any arithmetic expression can be enclosed in parentheses.
2. Arithmetic elements or expressions can be connected by arithmetic operators to form other arithmetic expressions, provided that no two arithmetic operators appear in sequence and no arithmetic operator is assumed to be present.
3. An arithmetic element or expression preceded by a sign (either + or -) is permitted, whereas the operators * and / must be enclosed by elements and/or expressions.
4. All constants in an arithmetic expression must be of the same type. Similarly, all variables in an arith-

metic expression must be of the same type. Mixed-mode arithmetic should not be attempted.

When division is performed in an integer arithmetic expression, the integral part of the quotient is retained and the fraction is discarded.

Logical Expressions

A logical expression is used in an **IF** phrase and can take any of the following forms:

1. A single logical variable.
2. Two or more logical variables, connected by the logical operators **&** or **|**, denoting logical **AND** and logical **OR**.
3. Two arithmetic expressions of the same type, connected by a relational operator.

The rules for constructing logical expressions are:

1. A logical expression that contains a relational operator has the logic value "true" if the condition expressed by the operator is met when the expression is evaluated. Otherwise, the expression has the value "false."
2. The \neg logical operator must be followed by a logical expression or term. Similarly, the operators **&** and **|** must be enclosed by logical expressions to form more complex expressions.
3. Any logical expression can be enclosed in parentheses. Any logical expression containing two or more variables to which the \neg operator is to apply *must* be enclosed in parentheses.

Logical expressions that do not contain parenthetical terms are evaluated in the following order: terms connected by relational operators are evaluated, then terms connected or modified by logical operators. When parenthetical terms are included in a logical expression, evaluation is performed in the order indicated above on the terms within parentheses, then on the reduced logical expression (reduced in that the parenthetical terms have been assigned a single logical value) in the same order. Logical expressions are evaluated by **PCS** in the same manner as **FORTRAN**. The major difference is the notation used for relational and logical operators, as illustrated in the following table:

FORTRAN OPERATOR	PCS OPERATOR
.GT.	>
.GE.	>=
.LT.	<
.LE.	<=
.EQ.	=
.NE.	≠
.NOT.	~
.AND.	&
.OR.	

Ranges

The **DISPLAY**, **SET**, and **DUMP** commands may have as an operand two internal symbols separated by a colon (:), thus indicating a range of variables and arrays to be displayed or dumped. For example:

```
DISPLAY MAIN.I: MAIN.A
```

Commands

The following section describes, for each **PCS** command, the format of the command and its operands, and some examples of how it might be used in checking out a program.

QUALIFY Command

The **QUALIFY** command allows the user to enter implicitly qualified internal symbols, which are all defined in a single source program, after he has issued this command with the appropriate defining program name.

Example:

```
QUALIFY FTNPGM
```

where **FTNPGM** is the name of the compiled module. If the compiled object module has been link-edited, both the compiled object module name and the link edit output module name must be specified. An **ISD** must be available in order to use this command. When an object module has been link-edited, an **ISD** must have been requested for the output object module in addition to the one requested at compile time. Only one **QUALIFY** is in effect at a time, and each successive one overrides the previous one.

AT Command

The **AT** command causes control to be passed to **PCS** when the named **FORTRAN** statement is reached but before it is executed. Other actions (if any) in the **PCS** statement are performed prior to resuming execution.

The system assigns a **PCS** statement number to each statement containing an **AT** command.

Each time the statement specified in the **AT** command is reached during execution, a standard output is presented to the user except where the statement contains an **IF** condition that is found to be false. This output includes the **FORTRAN** statement number at which the command became effective, certain program status information, and the **PCS** statement number. The program status information includes the virtual storage location of the instruction being executed and the settings of several internal indicators.

Unless the **PCS** statement containing the **AT** command also contains a **STOP**, execution of the user's program

is automatically resumed following the actions requested. Only CALL, DISPLAY, DUMP, IF, SET, GO, BRANCH, and STOP can follow an AT. BRANCH and STOP must be the last in the dynamic statement. A GO command is meaningless in an AT statement and will be ignored.

Two or more operands, each separated by commas, can be specified in an AT command, but each must be the statement number of an executable statement.

Given the following statements in a source program named MAIN1:

```

      IF (A) 10, 20, 30
      .
      .
10   X = 1
      GO TO 40
20   X = 2
      GO TO 40
30   X = 3
40   IF (A . GT. B) GO TO 50
      GO TO 60
      .
50   .
      .
60   STOP

```

The flow through the IF statements could be traced with the PCS command.

```

QUALIFY MAIN1
AT 10, 20, 30, 40(2), 50

```

The counter referred to by the special character % is incremented by one each time the executing program arrives at a statement designated in an AT command. The counter is incremented even when the AT command is included in a PCS statement that contains an IF phrase that is evaluated as false.

DISPLAY Command

The DISPLAY command is used to print the contents of specified variables or arrays. The format of the output is determined from the type and length of the data.

The DISPLAY can have a list of operands, each separated by a comma. In addition to simple variables, the following can be displayed:

Arrays — When the operand of a DISPLAY is an array name without a subscript, the entire array is printed.

Ranges — When the first and last variable names, separated by a colon, are specified, a series of arrays and variables can be displayed. The user must be aware of the storage locations assigned by the compiler, since the last variable to be displayed must be in a higher storage location than the first variable. The storage assignments made by the compiler can be determined from the storage map optionally selected at compilation time.

Control Section — When the name of a COMMON block is specified, the entire contents of the area is displayed. The entire PSECT containing all non-common variables can be displayed by specifying the PSECT name. If the name of the COMMON block or PSECT is qualified by the module name (i. e., specified as a qualified internal symbol), each variable or array in the area is identified and formatted according to its data type. When the name is not qualified by the object module name (i. e., the external symbol form is used), the contents of the area are displayed in hexadecimal. This method can be used if an ISD is not available for the object module, but, again, a storage map should be available for locating the variables in storage.

Module Name — When a module name is specified as an operand, a control section map is formatted. The map contains such information as: the name, location, and length of each control section; the version of the module and each control section; and the entry point and save area location of the module.

Dummy Arguments — Arguments to subprograms cannot be displayed until the subprogram has been entered and they have been replaced by the actual argument values.

If the user is operating in conversational mode, he can terminate the action of the DISPLAY command by pressing the attention button at his terminal.

If a FORTRAN subroutine named SUBR consists of the following statements:

```

SUBROUTINE SUM (A, N, M)
COMMON B, C, D
REAL*8 A(N, M)
DO 20 J = 1, M
DO 10 I = 1, N
10  C = C+A(I, J)
20  CONTINUE
RETURN
END

```

Then the following would be valid DISPLAY commands:

```

QUALIFY SUBR
DISPLAY C,I,J      to cause each variable in the list to be
                   formatted and printed.
DISPLAY A          to cause each element in the array to
                   be displayed.
                   This statement is equivalent to
                   DISPLAY A(1,1):A(N,M)
DISPLAY B:D        to cause the contents of B, C and D
                   from the unnamed COMMON to be
                   printed.

```

DUMP Command

The DUMP command is used to cause the contents of specified variables or arrays to be written in the PCSOUT data set.

The DUMP command provides exactly the same results as the DISPLAY command. DUMP should be used when there are large amounts of data and/or when an offline output of data is desired. The operand of the DUMP command is identical to that of the DISPLAY command.

The DDEF command must be used to define the PCSOUT data set before the DUMP command is issued. If no definition has been given, the conversational user is prompted to issue it. In a nonconversational task, if the DDEF is not entered prior to the DUMP, the task is terminated. The organization of the PCSOUT data set is VI.

The user is prompted at log-off time as to whether or not he wishes to catalog the data set. Since DUMP output is not interspersed with SYSOUT output, the user should provide a means of correlation if one is required.

Using the sample source program from the DISPLAY command description, the user might enter the following commands:

```
DDEF PCSOUT, VI, DSNAME=name
DUMP SUBR. %COM      This command would cause the
                     contents of the unnamed COM-
                     MON block (in this case, vari-
                     ables B, C, and D) to be format-
                     ted and placed in the PCSOUT
                     data set.

DUMP SUBR. SUBR#P    This command would cause the
                     contents of the subroutine's
                     PSECT to be formatted and
                     placed in the PCSOUT data set.
                     Included would be variables I
                     and J plus miscellaneous con-
                     stants generated by the compiler
                     as needed for program execution.
```

IF Command

The IF command is used to specify a logical expression that must be true in order for any other commands in the statement to be performed.

If the PCS statement containing the IF also contains one or more AT commands, the logical expression is evaluated when the statements specified in the AT command are reached. For example, the following statements appear in a source program named CALC:

```
5 DO 10 I = 1, 5000
  CALL SUB(ANS)
  IF (ANS) 20, 10, 10
10 CONTINUE
20 STOP
```

If it is discovered that the upper limit of the DO loop (5000) has been set too high, the following PCS statement could be entered.

```
QUALIFY CALC
AT 10;IF I=1000;STOP
```

Then, once execution starts, each time control passes to statement 10, the value of I is tested for equality to 1000. If I reaches the value of 1000 before going to 20, the PCS STOP command is performed.

The counter associated with a PCS statement containing the AT commands, referred to by the special character %, is incremented by one whether or not the logical expression in the IF is true.

The % counter can be useful in controlling loops in source programs and in controlling the effectiveness of PCS commands. For example, the above PCS statement could be replaced with:

```
QUALIFY CALC
AT 10;IF %=1000;STOP
```

In the last statement, rather than testing the value of the variable I, a count is incremented each time control passes to statement 10. The STOP is activated when the count reaches 1000.

Other uses of the % counter can be seen from the following examples:

```
QUALIFY CALC
AT 10;IF %=1;DISPLAY ANS
```

The DISPLAY would be performed only on the first arrival at statement 10

```
QUALIFY CALC
AT 10;IF %=(%/10)*10;DISPLAY ANS
```

The DISPLAY would be performed every tenth time statement 10 is reached. This example shows that the fraction is discarded in integer division.

REMOVE Command

The REMOVE command permanently cancels all dynamic PCS statements whose numbers are specified as operands. PCS produces a statement number following entry of each statement containing an AT command. These statement numbers are used in the REMOVE command to specify the PCS statements to be canceled.

Taking the following source statements as an example:

```
C PROGRAM MAIN
10 READ (1, 20)A
20 FORMAT (F6. 2)
  IF (A) 10, 30, 30
30 B = A*3.14
  WRITE (2, 20)A, B
  GO TO 10
```

then the following statements could be entered to check the progress through the program:

```
LOAD MAIN
QUALIFY MAIN
AT 10(2); DISPLAY A
AT 30(2); DISPLAY B; STOP
CALL
```

When control reaches the IF statement, the value of A is displayed; when the WRITE statement is reached, the value of B is displayed. The STOP causes the next command to be read. The user might then decide that the program is executing correctly, and wish to continue running without the checkout statements. He could then enter:

```
REMOVE 1, 2
GO
```

In this example, the numbers 1 and 2 are the PCS statement numbers that have been assigned to each AT command. They are printed immediately following entry of the statement either at the user's terminal if in conversational mode or on the SYSOUT data set if in nonconversational mode.

CALL, GO, BRANCH Commands

These commands can be used either as a separate command or as part of a PCS statement. The effect of these commands is to transfer system control of a task from command mode to program execution mode.

The commands have three forms: CALL, GO, BRANCH.

1. CALL [module-name]

The CALL command loads the module named in the operand (unless it is already loaded) and initiates execution at the beginning of the program. Only main programs should be referenced by the CALL command; otherwise, the results are unpredictable.

2. GO

There is no operand. This command is used when the user wishes to resume execution following a PCS STOP command, a FORTRAN PAUSE statement, or an attention interrupt.

3. BRANCH [module.stmnt-no]

This is used when the user wishes to change the flow of a program; it is equivalent to a GO TO statement in the original source program. The object module must be loaded and executing prior to issuing the BRANCH. This form cannot be used in initiation of program execution or in situations that are illegal in the FORTRAN language (e.g., illegal entry into loops). The statement number must be an executable statement.

For example, if the following statements appear in a source program:

```
C PROGRAM MAIN1
  :
  :
  :
10 X = A
   GO TO 20
20 X = B
```

```
30 CALL SUBR(X)
  :
  :
```

and the user discovers that the GO TO statement has the wrong statement number, he can temporarily correct the invalid GO TO with the following PCS statement, so that the rest of the program can be debugged.

```
QUALIFY MAIN1
AT 10(2);BRANCH 30
```

Note that the BRANCH command combined with an AT cannot be used to insert a missing GO TO statement without bypassing the statement referred to in the AT. In the above example, if the GO TO had been missing from the source program, the BRANCH command would have to be made effective at statement 10. In this case, the assignment statement $x = A$ would be bypassed, so that a SET command would be necessary to achieve the same results.

SET Command

The SET command is equivalent to an assignment statement in the original source program. It enables the user to change the contents of any variable or array element. It has the form:

```
SET a=b
```

where a is any simple variable or subscripted array element, and b is any logical or arithmetic expression. A list is allowed; for example:

```
SET a=b, c=d, e=f
```

When the SET is performed, the new value of the variable is displayed on SYSOUT in the same format as if the name had appeared in a DISPLAY command if LIMEN=1. This output is produced from the changed field and reflects the results of conversions and expression evaluation.

In a SET command, all variables and constants must be of the same type. The permissible lengths vary with the type. Real variables or constants must be 4 or 8 bytes in length, integer 2 or 4, and logical 1 or 4.

If the expression is a character constant that is not the same length as the variable, the character constant is either truncated or filled out on the right with blanks. If the expression is a hexadecimal constant that is not the same length as the variable, the hexadecimal constant is truncated or filled on the left with zeros.

In the example given for the CALL, GO, and BRANCH commands, the missing GO TO statement could be effectively inserted by the following PCS statement:

```
AT 10;SET X = A;BRANCH 30
```

The SET command is useful in setting variables to some initial value, in correcting erroneous assignment statements, and in inserting missing assignment statements. It should not be used in a situation that is invalid in the original FORTRAN program. For example, in the following statements:

```
5 DO 10 I=J, K
  L(I)=L(I)+M
10 CONTINUE
```

the values of I, J and K cannot be changed by a dynamic SET command that may be activated at statements 5(2) or 10. The SET command could be used at statement 5, however, to initialize the values of J and K. For example:

```
AT 5;SET J = 1, K = N/2
```

The SET command, like DISPLAY and DUMP, can refer to dummy arguments to a subprogram once the subprogram has been entered.

A complex variable can be SET to the value of another complex variable, but it cannot be SET to a constant value, nor can complex arithmetic be performed with SET. For example, if a FORTRAN program had the following specification statement

```
COMPLEX*16 A, B, C(10, 10)
```

the following SET commands would be valid:

```
SET A = B
SET C(1,1) = C(2, 1)
```

A SET command cannot refer to the control section containing instructions generated by the compiler. The virtual storage assigned to this control section by the system is protected so that its contents cannot be changed.

STOP Command

The STOP command halts execution of a module and prints the current instruction location and program status information. STOP does not have an operand. Execution of the module can be resumed with a RUN or GO command; if execution is not resumed, any data sets that the module has left open should be closed with the CLOSE command.

PCS Diagnostics

PCS, like the FORTRAN compiler, examines each statement for validity and issues diagnostics alerting the user to errors.

Diagnostics usually are issued immediately upon reading the command. The conversational user can reenter the statement with the necessary correction made. The nonconversational user has no chance to correct errors; a diagnostic message is issued and the PCS statement is ignored.

Certain errors are not detected until execution has begun. These errors are the result of some action that has been requested in a dynamic PCS statement (i.e., one containing an AT command). In a conversational task, after the diagnostic is issued, the terminal is placed in command mode. The user can then *remove* the erroneous statement, reenter it correctly if he desires, and continue execution with a GO. If he wishes to perform the corrected statement immediately, he must use the statement number in the AT operand of the BRANCH. In a nonconversational task, the diagnostic is written on the SYSOUT data set and the next command is read from SYSIN. This may result in prematurely terminating program execution.

The errors described below are those that are not detected until execution has begun.

Dimension Errors

Each time the user refers to a subscripted array in a PCS command, the subscript values are checked against the dimensions of the array as declared in the FORTRAN program. Since the values of variable subscripts may vary during execution, the error is not detected until the command containing the invalid reference is performed. Constant subscripts that are in error are detected when the statement is first read, and the user is informed immediately. If an array is a dummy argument, subscript errors are not detected until the command is performed, since both the dimensions of the array and the subscripts may be variable.

Range Errors

A DISPLAY or DUMP command may have as its operand a range of symbols. These symbols, which represent the starting and ending storage locations to be printed, must be in sequential order. When both symbols are subscripted arrays, the subscripts must be evaluated to determine the particular element being referred to. To illustrate:

```
QUALIFY MAIN1
AT 10;DISPLAY A(I):A(J)
```

If, when control reaches statement 10, the value of I is higher than the value of J, the range would be invalid.

Program Interruption

Program interruptions can occur any time an expression must be evaluated in a PCS command. Five such interruptions are recognized:

1. Fixed-point overflow exception
2. Fixed-point divide exception
3. Exponent-overflow exception

4. Exponent-underflow exception
5. Floating-point divide exception

When any of these interruptions occurs, a warning message is issued to the user and the requested action is not performed. These interruptions are *not* processed by the interruption handling module provided by the compiler; therefore, any `CALL OVERFL` or `CALL DVCHK` statements do not recognize the interruption. (Refer to Appendix C for more details on program interruptions.)

Dummy Arguments

When dummy arguments are referred to in a `PCS` command, the subprogram in which they are declared must be entered prior to the point where the command is activated. Dummy arguments must not be used to form a range of variables to be displayed or dumped. There are also those conditions under which dummy arguments are not defined in the `ISD` for the subprogram.

Appendix C. Programming Considerations

This appendix addresses a number of topics that can assist the TSS/360 FORTRAN programmer in achieving efficient and trouble-free execution of his object program. The sections of this appendix discuss:

1. Object-time efficiency through compiler optimization, optimal use of source statements, and use of the linkage editor and dynamic loader.
2. Effect of compiler optimization on the use of the program control system (PCS), describing conditions under which the user may want to inhibit optimization by the compiler of the object code.
3. Multiple executions, to alert the user to possible problems when executing more than one program between a LOGON and a LOGOFF.
4. Library management.
5. System naming rules that prevent the user from inadvertently choosing a subprogram or other external name such that a conflict would occur between this name and a system name.
6. Executing commands from within a FORTRAN program.
7. Miscellaneous programming considerations.

Object Time Efficiency

Object Code Optimization

Efficient object code can be achieved by optimizations performed by the compiler, by optimal ordering of source statements by the programmer, or both.

Compiler Optimization

This section describes optimization of the user's program normally performed by the compiler. The section "Effect of Compiler Optimization on PCS Usage" discusses the relation between compiler optimization and use of the program control system (PCS).

A considerable amount of the compiler's effort is devoted to producing an efficient object program. This processing is called "optimization." The effects of the compiler's optimization can be seen by examining the optional object code listing. It can be observed that the instructions to perform certain computations are sometimes not located where one would expect to find them. This is due to the action of two optimization processes: (1) recognition of "common" expressions and (2) removal of expressions from DO loops.

Two occurrences of the same expression in a FORTRAN program are "common" if there is no possibility of any of the operands receiving a new value between the occurrences and if program control cannot reach the second occurrence without having passed the first occurrence. In this situation, the compiler often generates code to evaluate the expression only at the first occurrence and to reuse this value at the second and later occurrences.

If an expression occurs in a DO loop and if none of its operands can have different values for different repetitions of the loop, the expression is "removable." The compiler generates code to evaluate such an expression before entering the loop and to use the computed value where needed inside the loop.

These processes can contribute much to the efficiency of the object program, but there is an important side effect. If the program is testing for such conditions as arithmetic overflow and divide check, the operations giving rise to these conditions may not occur at the expected place or with the expected frequency. A related optimization process is the computation by the compiler of all quantities whose operands are constants instead of the generation of instructions to carry out the computation in the object program. If the values of the constant operands are such as to cause overflow, the overflow will take place during compilation (causing a diagnostic message) rather than during execution of the object program.

Compiler optimization can also cause register contents to be used at points quite remote from the point of loading. In some cases, frequently used address constants may be loaded into general registers only at the beginning of the program and kept there permanently for use as needed. Other addresses, subscript expressions, etc., may be held in registers across the range of a DO loop or nest of loops. Even where such "global" register assignments are not made, the contents of any register, once established, is remembered and may, under the proper conditions, be used later without reloading.

Subscript expressions, especially those involving DO loop variables, receive extensive manipulation, and their evaluation may be spread over several levels of a DO loop nest. In such instances, an occurrence of a loop variable multiplied by other factors is evaluated by initialization at the top of the loop and addition of an increment at the bottom just before returning for another iteration.

If the most recent value of a DO loop variable is always stored in its assigned location each time through the loop, the DO loop variable is said to be "materialized." In many DO loops, there is no computational need for the value of the loop variable and it may not be materialized. Instead, a subscript expression involving the loop variable will be tested to determine the current number of repetitions of the loop. A feature of the FORTRAN language is that no assumption can be made about the value in the storage location assigned to a DO loop variable after the loop has been executed due to completion of the proper number of repetitions. (This does not apply to other exits from the loop, since the existence of such exits as a GO TO statement causes the compiler to materialize the loop variable.)

Examination of the optional storage map of the object program produced by the compiler can show that the storage assignments for non-COMMON variables are not made in the order of declaration or appearance in the source program. Rather, these variables are assigned by the compiler in an order intended to minimize the number of distinct address constants and subscripts needed in the object code. Undimensioned variables are placed first, followed by arrays in order of increasing size and dimensionality. Variables whose assignments are controlled by EQUIVALENCE relations are placed after those that are not in EQUIVALENCE.

The purpose of EQUIVALENCE is to permit overlays to reduce object-time storage use. It is not intended to permit intermingled references to the same storage locations by two different names. If X and Y are assigned to the same storage location by EQUIVALENCE, and a value is given to X, there is no guarantee that a subsequent reference to Y in the same program will use this value. The compiler's optimization processes do not recognize the relationship between X and Y.

Efficient Use of FORTRAN Statements

The above section described optimization performed by the *compiler* on FORTRAN programs. Further optimization of the object code can sometimes be achieved by the FORTRAN user's being aware of uses of the FORTRAN language leading to more efficient object code. Such uses are described here. Under no circumstances, however, is the user required to program in accordance with the guidelines presented here.

The compiler's optimization is limited in various ways, such as lack of freedom under the rules of the FORTRAN language (e.g., the compiler cannot rearrange variables in a COMMON block) or lack of information (e.g., the compiler cannot make any assumptions about the behavior of external subprograms).

In laying out user-controlled storage (COMMON and EQUIVALENCE), the user can find it worthwhile to align the storage locations for all variables to the proper byte boundaries for their arithmetic or logical type: doubleword, fullword, or halfword quantities should be assigned to locations that are multiples of eight, four, or two storage locations, respectively, from the beginning of the area being laid out. This permits access to these quantities by machine load and store instructions rather than by subroutines accessed as the result of specification exceptions. (See the discussion of object program interruption provisions in the Miscellaneous Programming Considerations section below.)

If the user, when ordering a COMMON block, follows the same criteria used by the compiler in laying out non-COMMON variable storage, the same benefits accrue. Placing scalars first, then small arrays, etc., tends to improve the address coverage in the object code and obviates the need for object-time boundary alignment.

Since each COMMON block must be covered by its own address constants, the use of a large number of small COMMON blocks leads to less efficient addressing than a few large blocks.

In passing information to a subprogram, explicit arguments are more expensive than implied arguments in COMMON. Not only must each individual argument have its own address cover in a subprogram, but instructions in the subprogram prologue must be executed to move the address in from the calling sequence. A group of arguments in a COMMON block, however, can all be addressed with the same address constant and need no initialization.

The compiler's manipulation of expressions is restricted by the FORTRAN language requirement that the source program associations (both explicit associations determined by parentheses and those implied by left-to-right order) be respected. $A+B+C$ must be treated by the compiler as $(A+B)+C$. Common expressions, removable expressions, and constant expressions can be recognized only if the associativity permits. Therefore, the efficiency-minded user writes expressions in such a way as to permit optimization. For example, in $A*(2./3.)$, the division of constants is done by the compiler; in $(A*2.)/3.$ it is not. If K_1 and K_2 are unchanging in a DO loop on I, the sum K_1+K_2 is computed outside the loop if K_1+K_2+I occurs; it is not removed from $I+K_1+K_2$, or K_1+I+K_2 .

The compiler is unaware of the properties of subprograms external to a program being compiled. Even for FORTRAN IV-supplied subprograms such as SIN, the user is free to substitute his own subprogram for the library routine. Therefore, the compiler does no op-

timization on external function calls. If the user wants to save execution time by eliminating redundant calls on the same function with the same arguments, he must do this himself in his FORTRAN program. Only the user has the necessary knowledge of whether or not a function uses or changes variables in COMMON, changes its arguments, performs I/O, runs internal counters, etc. (The function routines supplied in the FORTRAN IV-supplied subprograms do none of these things.)

The use of mixed arithmetic types in expressions and across the equal sign in assignment statements leads to the execution of conversion functions, the most expensive of which are those converting REAL to INTEGER and vice versa. Unnecessary use of mixed types thus diminishes efficiency.

Although the compiler may remove expressions from DO loops, it does not remove complete statements. For example, every assignment statement occurring in a loop results in at least a storing of a value into the left-side variable on each repetition of the loop. For efficiency, statements that are entirely invariant within a loop should not be placed inside the loop.

Much normal optimization is inhibited for DO loops with extended range (i.e., a branch out of the loop to execute some remote statements, followed by a branch back into the loop). This programming practice reduces the efficiency of those DO loops in which it is used.

Computed GO TO statements with three or less destinations produce less efficient code than equivalent arithmetic IF statements.

At the time of writing his program, a user thinks in terms of virtual storage, of which he has a very large amount available. During execution of the program, the system maps this virtual storage onto the much smaller actual main storage of the computer, using a page (4096 storage locations) as the basic unit. Each page of virtual storage referred to in the program must be made available in main storage before the reference can be successfully made. Two attributes of a program can decrease the efficiency of rts/360 in carrying out this task, resulting in a loss of system performance. These attributes are references to a number of different pages in rapid succession and a large number of total pages required. Some simple programming and operating practices can alleviate both problems.

When indexing a large multi-dimensional array, it is better to vary the left-most subscript the most rapidly. This causes the array elements to be accessed in order of their location in storage rather than out of order. For example:

```
DO 24 J=1, 50
DO 24 I=1, 80
24 A(I, J)=1. 0
```

is better than

```
DO 24 I=1, 80
DO 24 J=1, 50
24 A(I, J)=1. 0
```

from the point of view of page utilization.

In coding a subprogram, it may be worthwhile to move the value of an argument into the subprogram by setting a local variable equal to it, if the argument is referenced frequently but is not an output argument of the subprogram. For example:

```
FUNCTION F(X)
XLOCAL=X
.
.
.
Y=XLOCAL/Z
.
.
.
```

(and other references to XLOCAL, rather than X)

is better than

```
FUNCTION F(X)
.
.
.
Y=X/Z
.
.
.
```

(and other references to X)

since in the latter case the actual argument presented for X in a call on the subprogram is located in some other module and, hence, probably on a page that would otherwise not need to be accessed during the execution of the program F.

The user can also minimize the number of pages referred to, where the executable code generated for his FORTRAN statements exceeds one page or extends over many pages. The general rule for attaining minimum page references for such programs is to place infrequently used statements in a separate subprogram. If this is inconvenient, the infrequently-used statements could be placed in a separate area of the program, near the end perhaps, while grouping together those areas of the program that will be executed most frequently. Since calls on subprograms are in general references to separate pages (unless link editing is performed, as discussed below), it may be more efficient to minimize the number of subprogram calls within areas where optimum efficiency is desirable.

Use of Linkage Editor to Improve Object-Time Efficiency

Linkage editing is a valuable process for reducing the total number of pages required for execution of a pro-

gram using many subprograms and for obtaining better utilization of allocated storage. Object program loading time can also be reduced by linkage editing to package the control sections together and thus produce a dense packing of virtual storage. The order in which programs are packed should be such that as few references as possible are made by code in one page to code in other pages. Maximum efficiency can therefore be achieved if the packing is done following a study of relation between programs and between different parts of large programs.

Use of Dynamic Loader to Improve Object-Time Efficiency

Explicit unloading (using the UNLOAD command) of modules that are no longer of interest to the user in a session is a good practice if:

1. these modules referred to a great many different pages of virtual storage; and
2. no further references will be made to these pages.

Use of Control Section Packing to Improve Object-Time Efficiency

To allow the system to function more efficiently when executing object program modules, a dynamic method is provided for combining more than one control section into a single page of virtual storage at *execution time*. Fewer pages will thus be referred to, reducing system paging requirements. Control sections of like attributes within a module will be allocated contiguous storage, with the secondary control sections aligned on doubleword boundaries. You elect the type of control section packing to be used (if any) in your LOGON command.

Effect of Compiler Optimization on PCS Usage

Appendix B contains a description of the use of PCS with FORTRAN programs. In Appendix B it is stated that the FORTRAN user should select the ISD option to ensure that the PCS statements operate correctly. The following paragraphs explain why the PCS user should generally request an ISD, and under what conditions he need not request an ISD.

Due to the optimization method, use of PCS with optimized FORTRAN programs might easily lead to erroneous results. For example, consider the following program:

```

50 DO 200 I=1, N
100 X(I)=A(I) + (B+1.0)
200 CONTINUE
.
.
.
```

The sum of (B+1.0) is in no way affected by the fact that a loop is occurring in which the sum is required, so the compiler computes the sum once, outside the loop, and uses this value inside the loop. Suppose now that the user wants to change the value of B at statement 100 using the PCS SET statement. A fully optimized object program would not be aware that B has been reset, as (B+1.0) would not be recomputed within the loop; thus, the user's intent would not be accomplished.

A second example of compiler optimization leading to problems when PCS is used would be where, in the above example, the FORTRAN user wants to use the PCS DISPLAY command to determine the current value of the loop variable I. The object code would have no need for I except for counting passes through the loop and indexing into the X and A arrays, so no code would be generated to save the current value of I in the storage location assigned to I. (Indeed, no storage location may have been assigned to I.) Thus a DISPLAY of I would not produce the desired results.

It is clear that such complex restrictions on PCS usage as implied by the compiler optimization procedures would not be desirable. For this reason, the FORTRAN user may request that the compiler modify its generation of the object program in such a manner as to allow complete use of PCS facilities on FORTRAN object programs. The user requests such modification of object code generation by specifying, following his entrance of the FTN command, that he wishes an Internal Symbol Dictionary (ISD). Such a request:

1. Inhibits optimization as required to allow all PCS capabilities to be available
2. Produces an ISD, which allows the FORTRAN user easy reference to all symbols within the FORTRAN subprogram

A user can still use PCS, of course, even if he does not request an ISD. To do so, however, he must have extensive knowledge of his object code to be assured that his PCS requests will give the desired result.

There is one type of PCS usage in which the FORTRAN user must exercise considerable care, even if an ISD is requested. This use of PCS can be described with the aid of the following program:

```

.
.
.
DO 10 I=1, M
M=M+1
5 L(I)=L(I)+M
10 CONTINUE
```

The statement M=M+1 is clearly illegal, as it violates a FORTRAN language rule (and the compiler produces a diagnostic message). Just as it is illegal in

the original FORTRAN program, it should not be simulated by compiling a program without the $M=M+1$ statement but directing PCS to add one to M at statement 5. Similarly, in the following example, the use of PCS to specify that at statement 100 the program should transfer control inside the DO 200 loop (using the PCS RUN directive) is not legal, as this would violate a language rule were it replaced by a GO TO 200 in the original FORTRAN program.

```

      .
      .
100  .
      .
      DO 200 I=2, 10
      A(I)=A(I+1)/A(I-1)
200  CONTINUE

```

However, it is legitimate in the following program, at statement 100, to direct the object code to RUN at statement 200, as this is equivalent to a GO TO 200 at source statement 100, a legitimate branch.

```

50  READ (5, 9, END=200) A, B
     X=A**2+A/3.1416-A**B
100 WRITE (6, 9)A, B
     GO TO 50
200 STOP
     9  FORMAT (2E20, 7)
     END

```

Multiple Executions

“Multiple executions” refers to executing more than one program between logging on and logging off.

Data Definition Considerations

A DDEF command provides the linkage between the data set reference number used in the FORTRAN program and the actual data set. Once a DDEF has been entered, it remains in effect until LOGOFF unless the definition is released or redefined.

If two programs are executed in succession, the following conditions could arise:

1. Both programs refer to the same data set with the same data set reference number. One DDEF command issued prior to the execution of the first program is sufficient for both executions if the data set is read in both programs or written in the first and read in the second. If, however, the data set is written in both programs and not rewound in the first, the data is not automatically concatenated. Data written in the first execution would be destroyed by the write operation in the second execution. If the user does not want this to occur, he must take the steps outlined in item 3.

2. Both programs refer to the same data set with different data set reference numbers. Each execution must be preceded by a DDEF command giving the ddname as appropriate for the data set reference number. Since the second DDEF contains the same dsname as the first, effectively redefining it, the first definition need not be released.
3. Each program refers to a different data set with the same data set reference number. Each execution must be preceded by a DDEF command giving the dsname for the ddname. In addition, since the second DDEF has the same ddname, the first definition must be released prior to the second DDEF. When a data set on a private volume is released, the input/output device is also released unless another defined data set resides on that same volume. In a nonconversational task, if a device is freed by a RELEASE command, the user must account for this when specifying the SECURE command. For example, if two programs read different data sets on separate private volumes and both are referred to by data set reference number 1, the following procedure is necessary:

- | | |
|------------|---|
| a. SECURE | Two devices — even though only one device is needed at any one time |
| b. DDEF | For first data set |
| c. CALL | First execution |
| d. RELEASE | First data set |
| e. DDEF | For second data set |
| f. RUN | Second execution |

Linking COMMON Between Multiple Executions

When executing a series of programs in sequence, if a prior module is not unloaded before the execution of the next program, any external symbol reference in the second module will be resolved, if possible, by definitions of that symbol in the first module. This may or may not be desirable. If this is not desirable, an UNLOAD command should be issued after the first main program has completed execution, causing any blank or named COMMONs to be removed from the task's allocated storage. Any subsequent module that is loaded containing a COMMON block would have storage allocated as if it were the first usage.

When the user does want to pass the same COMMON block from one execution to the next, the UNLOAD command should not be entered. In this case, the references to COMMON in the second execution would be to the COMMON that was allocated storage with the first execution, if both are unnamed or have the same name.

Program Libraries

Program Library List Control

A program in TSS/360 can consist of one or more object modules. All programs in TSS/360 are stored in object module form in program libraries, which are partitioned data sets. A program consisting of only one object module is stored entirely within one library; however, if a program consists of several object modules, those modules may reside in different libraries, depending on how the user has stored them. During link editing and execution, the system can automatically retrieve all object modules required, if the user has defined the libraries containing those object modules. The method for doing this is described in the following paragraphs.

There are four categories of program libraries:

1. System library (SYSLIB)
2. User library (USERLIB)
3. User-defined job libraries
4. Other user-defined libraries

TSS/360 does not allow a library to contain more than one declaration of *any* external symbol, except those control sections that have no content (e.g., named or blank COMMON from a FORTRAN main or subprogram).

The system library contains some service routines provided by the installation. It also includes the FORTRAN supplied subprograms.

The user library is the private library assigned to each user when he is joined to the system. This library is automatically defined for him and made a part of his catalog by the system. His user library is thus available each time he logs on. If the user does not specify job libraries in a task, the object modules resulting from his use of the language processors are placed in his user library.

The user may want to restrict his user library to object modules that he executes frequently or that he uses frequently in the buildup of other object modules. The system's library list facilities make it possible for the user to control the contents of his user library.

The program library list is a defined hierarchy of program libraries. It is initialized at log-on time and consists of the user library and SYSLIB.

The library at the top of the list automatically receives all object modules resulting from language processing. As noted above, if no job libraries are defined, the library at the top of the list is always the user library. However, the user can specify that a job library be added to the program library list to receive the output of the language processors. He does this by issuing a DDEF command defining that

job library and containing the operand OPTION= JOBLIB. When this command is executed, the name of that job library is added to the top of the program library list. That library then receives all subsequent module output of the language processors until another job library is defined (and is thus at the top of the list) or until a RELEASE command is issued for that job library. A job library must always have a VP data set organization; it can be defined on public or private volumes.

In addition to using the program library list to store object modules, the system uses this list to control its order of search when looking for object modules that must be loaded at execution time. The library at the top of the list is searched first, then the next-to-the-top library, etc. until finally, the user library and SYSLIB are searched.

In summary, the user has the following basic library setups for handling the object modules by the language processors.

- User library — As this is always available and is always searched, the user may want to reserve it for frequently used checked out programs. All user's USERLIBS are kept in public storage and, hence, are always mounted on system devices.
- Session JOBLIB — By issuing a DDEF command for a new library at the beginning of a session, a user can create a library to contain all modules assembled or compiled during the session.
- Cataloged Private Volume JOBLIB — A user can direct output to and retrieve it from a library of infrequently used modules by issuing a DDEF command for a cataloged job library that resides on a private removable disk pack. When using private job libraries in a nonconversational task, the user must request (via SECURE) a device for that job library. Modules can be entered in such a library:
 - Automatically if the library is the latest defined one in the session.
 - By link-editing it from his USERLIB, session job library, or public device job library and specifying to the linkage editor the desired private device job library as the output destination.
 - Cataloged libraries on private volumes can also be shared among users.
- Cataloged Public Volume JOBLIB — This type of library can be useful in setting up (and using) a library of frequently used programs whose names and external symbols conflict with other programs in USERLIB. For example, versions of frequently used programs can be set up with one USERLIB and another in a job library. All job libraries re-

siding on public volumes are automatically cataloged at DDEF time and may be shared among users.

During linkage editing, the program library list can also be used to define to the system:

- The library that is to receive the link-edited object module.
- * The sequence in which libraries are to be searched if the system must find other object modules to define references in the linkage-edited object module.

The fourth category of libraries may be defined by a DDEF command with the JOBLIB operand omitted. Such libraries may be referred to by a linkage editor INCLUDE statement, but are not listed in the program library list, and hence are not included in the automatic library search, nor are they available to the dynamic loader. Refer to *Linkage Editor* for an explanation of linkage editor program libraries.

Since one library may not contain more than one definition of any external symbol, different versions of the same program must be kept in different libraries. For example, a user has a checked-out program in his USERLIB and wants to recompile the program with modification but retain his original version until the new version has been checked out. A DDEF with a JOBLIB option causes the new module to be stored on the job library rather than USERLIB. The user can continue after compilation with his checkout of the new version, since any subsequent LOAD or RUN command in the task naming the module retrieves the new version from the job library. If, when the new version has been successfully tested, the user wants to replace the old version with the new version, he can linkage-edit the new version onto his USERLIB. Link-editing can be used to copy a program module from one library to another. If the user does not want to retain the new version, he must either erase the module on the job library or release the job library. Releasing the library removes it from the program library list and automatically causes subsequent retrievals of that module to revert to USERLIB. Erasing the module does not remove the job library from the program library list, but any subsequent references to that module are resolved from USERLIB after the job library has been searched unsuccessfully.

The POD? command facilitates the orderly maintenance of programs within various job libraries and USERLIB. POD? enables the user to obtain on SYSOUT a list of the member names (and optionally the alias names and other member-oriented data) of individual members of cataloged VPAM data sets.

Substituting FORTRAN IV-Supplied Subprograms

All IBM-supplied subprograms are stored on the system library (SYSLIB). Any subprogram can be effectively replaced by storing a user's version (with the same name or entry point) on one of his own libraries, since the system library is always the last one searched. If the user stores his version on his user library, then all of the user's programs, when executed, refer to his version of the subprogram. If the user wants to refer selectively to either version, he should store his version on a job library, so that it is selected in a given task only if he has issued a DDEF for that job library.

When object modules are loaded by the dynamic loader, any substitute modules should be explicitly loaded (using the LOAD command) prior to issuing a RUN command for the main program. This guarantees that the desired modules will be used.

Note that, if a module is loaded explicitly, it will not be unloaded when the calling module is unloaded, i.e., it must be unloaded explicitly using the UNLOAD command.

When object modules are link-edited, references to modules on the system library are left for dynamic linking at LOAD time. If, however, the user has his own version of a subprogram on a job library in the program library list during the linkage edit run, this version is automatically included as part of his output module. He cannot then attempt to select either version of the subprogram during subsequent executions.

Sharing Libraries

A user can allow another user to share (i.e., access) one or more of his cataloged job libraries. When the owner permits access to his job library, all of the object modules on that data set are usable by the sharer. This does not imply that if the owner and/or one or more sharers use the same program at the same time they are sharing (co-using) the same copy in main storage. This aspect is controlled by the public option declaration at compile time.

The data set owner issues a PERMIT command to designate the other users who can share his job library and to indicate the level of access for those users.

- Read-only access — the sharer can use the object modules on the library but cannot add, replace, or erase a module.
- Read-and-write access — the sharer can use any object module on the library and can add or replace modules. He cannot use the ERASE command to delete a module from the library.
- Unlimited access — the sharer, in effect, can treat the library as his own; thus he can even erase modules from it. However, when any user with un-

limited access, including the sharer, attempts to erase a shared VI or VP data set, the system will first check to see if there are any active users of that data set. If there are active users, the system issues a diagnostic and disregards the command. If there are no active users, the system executes the ERASE command.

To gain access to a data set for which he has been previously authorized, the sharer must issue a SHARE command, which places an entry for the owner's data set name in the sharer's catalog. The sharer can then enter a DDEF command for the data set (with the JOB-

Table 6. Shared Data Set Commands

	BY OWNER	BY SHARER
PERMIT	Must be issued prior to the SHARE command by the sharers.	Not allowed. A user cannot permit access to a data set that he does not own.
SHARE	Not allowed.	Must be issued prior to any other references to the data sets. Once issued, the sharer may access the data set until he issues an ERASE or DELETE. The SHARE command places an entry in the sharer's catalog, so that a further CATALOG command is not necessary.
ERASE	The owner can only erase a member (object module) from his job library or erase the entire library when no sharer is accessing that member at the time the ERASE command is issued. If he erases the job library, the entry in the sharer's catalog is not removed. The sharers must issue a DELETE command to remove the entry from their own catalog.	A sharer can erase only if he has been granted unlimited access. If he then erases an object-module, neither the sharer's or owner's catalog is affected. If he erases the entire job library, both his catalog entry and the owner's are removed.
DELETE	The owner can delete a library or group of libraries from his catalog. An object module alone cannot be deleted. When the owner deletes a shared job library the sharer's catalog entry is not removed.	A sharer can delete his catalog entry for a job library without affecting the owner's catalog. The sharer must reissue a SHARE command if he again wants to refer to the data set that has been deleted.

Table 6. Shared Data Set Commands (continued)

	BY OWNER	BY SHARER
CATALOG	The owner can catalog a fully qualified data set name. If that name is a component of a partially qualified name that the owner has permitted to be shared, all sharers have immediate access to the newly cataloged data set. If an owner changes the name of a single data set to which he permitted access using a fully qualified name, each sharer must delete his catalog entry and reissue the SHARE command with the owner's new name.	A sharer that has been granted unlimited access can change or add entries to the owner's catalog. If he is permitted to share a group of data sets, he can catalog a new data set into the group, but he must include as part of the name the partially qualified name that he used in the SHARE command. If he changes the name of one of the data sets in the group, the new name must still contain the partially qualified name. A sharer who has been granted unlimited access to an individual data set can never change the data set name.

LIB option) in each task where he wants to include the library in his program library list.

Groups of job libraries with names having common higher-order components can be specified by using partially qualified names when the PERMIT is issued. For example, an owner of two job libraries named TRACK.SUB1 and TRACK.SUB2 can allow sharing of both libraries by using the partially qualified name TRACK in the PERMIT command. In this case, the sharer *must also* use the partially qualified name (as the dsname₂ parameter) in the SHARE command, even though he wants to access only one of the job libraries.

Table 6 lists the commands applicable to shared data sets and the effect of the command on the user's catalog.

Recovering from Errors When Dynamically Loading

The dynamic loader takes all of the external references in a module that is explicitly loaded or run and resolves them by searching the program library list. While the loader is linking the object modules into the user's virtual storage, diagnostics may be issued indicating any of several error conditions that can affect the eventual execution of that program.

- Name to be loaded or run not found in library — Either the user has specified the wrong name in the LOAD or CALL command or the job library containing

the object module has not been defined in the task and, therefore, is not in the program library list. In the latter case, the conversational user can merely enter the `DDEF` defining the job library and reissue the command.

- **Unresolved references** — If an object module refers to a `FUNCTION` or `SUBROUTINE` that cannot be located in any of the libraries in the program library list, a diagnostic is issued specifying the name that was used in the reference. Further linking of other object modules is not suspended, however, so that the main program and possibly other subprograms have been placed in the user's virtual storage. If the error occurs in a `RUN` command, execution of the program is not initiated.

If the user wants to execute his program regardless of the error, he can reissue the `CALL` command. He must, however, repeat the name of the module specified in the original `CALL` command. This is necessary to define the point at which execution is to be initiated.

If the user anticipates that an object module will have unresolved references, he should first issue a `LOAD` command naming the module, followed by a `CALL` command with the same operand. This procedure is recommended for a nonconversational task, since the user can be assured that execution will be initiated regardless of unresolved references.

If the user does not want to `CALL` the version of the program that has been put into his storage, he must issue an `UNLOAD` command. If he wants, he can then enter a `DDEF` defining a job library that was missing in the first `LOAD` attempt. A `LOAD` or `CALL` issued at this point causes the entire linking procedure to be redone.

- **Duplicate entry points** — This condition can only occur when two or more object modules are being linked from different libraries. For example, a user might `LOAD` a main program from a library which calls a subroutine on another library. If in this case, the subroutine had an `ENTRY` statement which duplicated the main program name or an `ENTRY` in the main program, a diagnostic would be issued indicating the error and the duplicated entry point name. The second entry point is disregarded by the loader, so that as the loading process continues all references to that entry point are resolved by the first definition — in this case, the definition in the main program rather than the subprogram. Subsequent execution of the program could give erroneous results if the references were incorrectly resolved. The user should take some corrective measures before attempting to `LOAD` or `CALL` again. (A possible

correction might be to change the `ENTRY` name by linkage-editing the object modules onto another job library. To avoid the possibility of such duplications when working with a new library, the `ROD?` command can be used to list the directory of the library. The user can then circumvent the problem by setting up an appropriate program library list before he attempts to load his program.

Shared Code (PUBLIC) Considerations

The system recognizes a control section as being either private or sharable. Normally, both the `PSECT` and the `CSECT` of the output module are marked by the compiler as private. However, if the public option is selected in the `FTN` parameters, the `CSECT` is marked as public. If the library containing such a module is a shared data set (i.e., `PERMIT` and `SHARE` commands have been issued), the `CSECT` is considered sharable.

Each task is allocated its own copy of a private `CSECT`; however, allocation of public `CSECTS` occurs in such a way as to make the same physical copy of the `CSECT` available to all tasks which have allocated the `CSECT` to their respective virtual storages.

Sharing object code enhances the efficiency of the system. Paging is reduced since only one copy need be in main storage or on the paging device; in addition, shared routines can be executed simultaneously by more than one CPU.

Any user-written `FORTRAN` program can be made sharable by specifying the `PUBLIC` option in the `FTN` parameters.

Prior to compiling the module, a `DDEF` must be issued defining the job library where the object module is to be stored. Once the module is compiled, the user must grant access to the job library by issuing a `PERMIT`. This, of course, is not necessary if the object module is stored on a job library previously being shared.

Each user who has been permitted access must then issue a `SHARE` command, to make the appropriate entry in his catalog for the library. Again, this is not necessary if the user is already sharing the data set. Each time the sharer wants to use the shared program, he must issue a `DDEF` for the job library prior to loading the object module. The object code is only truly shared (only one copy in main storage) when each user loads the public control section from the same shared job library. A sharer who linkage-edits a public control section onto another library receives a private copy each time the object module is loaded from that library.

The owner of a data set may `PERMIT` any level of access he wants regardless of the access designator

in the owner catalog. For example, if the owner catalog is marked "read only," the owner may not write into his own data set, but he may PERMIT a higher level access (read/write or unlimited) to a sharing user. Because of this flexibility, the data set owner should be very cautious with critical data sets that he has entered into the system.

Note: A program requiring more than 256 shared pages of storage cannot be loaded in public storage. The program will instead be loaded on private pages, and each user sharing it will receive a private copy.

System Naming Rules

User-Assigned Names

The following names resulting in external symbols are supplied by the user in his FORTRAN source program or during compilation:

- Module name (required)
- SUBROUTINE subprogram name
- FUNCTION subprogram name
- ENTRY names in subprograms (optional)
- Names of labeled COMMON

Names resulting in external symbols that are assigned in any single compilation must be distinct from each other. In addition, since the system does not allow any one library to contain more than one definition of a particular external symbol, each name (except names of COMMON blocks) must be distinct from any other symbol contained on the library that is going to receive the object module. It is valid to have the same names on different libraries. Since a named or blank COMMON is not listed in the directory of the library as an external symbol associated with this module, the name of the COMMON area does not have the preceding restriction. Also, since this name is not listed in the directory, it cannot be explicitly referred to by name (i.e., it cannot be loaded by its COMMON name).

The POD? command can be used to list external symbols in a library, thus assisting the user in avoiding duplication.

Reserved Names

External Symbols

The user can never assign an external name beginning with the characters `sys`. Names beginning with these letters are reserved for certain system programs. Any module stored on the user library or a job library starting with these symbols can never be retrieved by that name for execution, since resolution of `sys` symbols for loading and running is always attempted from the

system library. In addition, a diagnostic is issued if a module loaded by another name contains an external symbol beginning with `sys`.

The user should be careful to avoid accidentally duplicating the names of IBM-supplied subprograms. Generally, he should avoid the use of all external symbols starting with the characters `CHC` or any FORTRAN-supplied subprogram entry point name (i.e., `SIN`, `COS`, etc.) unless he specifically wants to substitute for such a subprogram one of his own.

Reserved Names Associated with Data Sets

The following list contains the reserved names that are assigned to system functions:

RESERVED DDNAMES	RESERVED DS NAMES
SYSLIB	USERLIB
SYSULIB	SYSLIB
SYSIN	
SYSOUT	
PCSOUT	

Compiler-Assigned Names

For each FORTRAN program compiled, the compiler makes the following name assignments resulting in external symbols:

- CSECT name — The module name (truncated to six characters if greater than six characters) is suffixed with `#C` to form the CSECT name.
- PSECT name — The module name (truncated to six characters if greater than six characters) is suffixed with `#P` to form the PSECT name.

NOTE: Since a BLOCK DATA program has neither executable instructions nor program variables, a CSECT or PSECT name is not assigned.

It is the user's responsibility to ensure that the first six characters of the module name are unique from others on the library receiving the object module. Since characters following the first six are truncated to form the CSECT and PSECT names, FORTRAN user may want to follow the practice of limiting the module names of compiled programs to six characters.

In addition to the CSECT and PSECT names, the following names are assigned to the compiler output data sets:

`SOURCE.module` — is the data set name assigned to the line data set of source statements constructed during the compilation. For example, a source program with module name `COWBOY` will be assigned the data set name `SOURCE.COWBOY`. If the input to the compilations was from a prestored data set, then the user must assign the name `SOURCE.module` to the data set prior to the FTN command.

LIST.module — is the data set name assigned to the data set created for all listings optionally selected by the user. Note that this is the index name of a generation data group.

Executing Commands from Within a FORTRAN Program

You can execute a command from within a FORTRAN program by calling module CGCDB at entry point SYSOBF. (CGCDB is a nonprivileged module residing in SYSLIB.)

Code the CALL statement with these three arguments:

First argument: The command length. (The number of characters in the command, not including the scratch byte in the second argument. Count double apostrophes — like those in the example below — as one character.)

Second argument: The command itself. Place this argument in single apostrophes. The first character of this argument must be a scratch byte that should not be counted in the command length.

Third argument: The name of the area in your program where you want the return code placed.

For example:

```
CALL SYSOBF (26,'_DISPLAY"AT STATEMENT 500"',IRC)
```

In this example, an underscore is used as the scratch byte.

Miscellaneous Programming Considerations

Floating-Point Computations

It must be kept in mind that, unlike integer arithmetic, floating-point computations (types REAL and COMPLEX) are not in general exact, due to roundoff, which may cause the low-order bits of a result to be different from the expected value. This consideration is especially important when writing FORTRAN relations or arithmetic IFs; exact equality of two floating-point quantities which are the results of computation is not to be expected. For example, consider:

```
Y = 0.1 * X  
IF (10.0 * Y - X) 1, 2, 3
```

The zero branch to statement number 2 will probably never be taken, since roundoff in the two multiplications, and the fact that 0.1 cannot be represented exactly in a binary computer, will cause at least the low-order bit of $10.0 * Y$ to be different from that of X , so the subtraction does not give a true zero.

Object Program Interrupt Provisions

This section contains descriptions of the procedures followed when the user's program is temporarily in-

terrupted due to certain types of interruptions. An interruption is a computer-originated break in the flow of processing. Program interruptions are those resulting from improper specification or use of instructions and data. The term "exception" is used to refer to these types of interruptions.¹ Six such exceptions occur frequently enough during normal FORTRAN programming to warrant special treatment. These are:

1. Fixed-point overflow exception
2. Significance exception
3. Exponent-overflow exception
4. Exponent-underflow exception
5. Floating-point divide exception
6. Specification exception

The procedure for handling the above exceptions is as follows. The compiler generates code at the beginning of all *main programs* that calls an interruption-handling module.² In this module the following operations are performed:

1. Initialization is performed such that the fixed-point overflow and significance exceptions will be ignored.
2. Initialization is performed such that a control will be passed to an entry in an interruption-handling module if any of the following four exceptions occur:
 - a. Exponent overflow
 - b. Exponent underflow
 - c. Floating-point divide
 - d. Specification

At the first three of these entries, flags are set for later interrogation by programs called as a result of the CALL OVERFL (if exponent overflow or underflow occurred) and CALL DVCHK (if divide check occurred) statements. The contents of registers following an overflow or underflow is:

Exponent Overflow: The sign of the result is unpredictable and the result characteristic is set to 127. In short precision, the high-order 24 bits of the fraction are set to one, leaving the low-order 32 bits unchanged. In long precision, all 56 bits of the fraction are set to one.

Exponent Underflow: The sign, characteristic and fraction of the result are set to zero, yielding a true-zero result. (In short precision, the contents of the low-order 32 bit positions of the fraction remain unchanged.)

¹For more detailed information, see *IBM System/360 Principles of Operation*, Form A22-6821.

²For a more detailed description, see *FORTRAN IV Library Subprograms*.

A specification exception will occur when a variable is not on a proper word boundary. This condition may exist in a FORTRAN program by forcing such misalignment through the use of a COMMON and/or EQUIVALENCE statement. The compiler issues a warning diagnostic if the user has forced such a misalignment, but such a misalignment does not prevent him from executing the program. An installation option specifies that one of two courses of action is to be taken if a specification interruption occurs:

1. Terminate the task.
2. Transfer control to a program that will perform the desired operation, using instructions that will not cause an exception due to the incorrect boundary alignment, followed by continuation of the user's program. This procedure is extremely inefficient, and thus should be employed as little as possible.

STOP/PAUSE/RETURN Differences

Table 7 summarizes the use of the FORTRAN STOP, PAUSE, and RETURN statements in both conversational and nonconversational mode.

Link-Editing FORTRAN Programs

The standard entry point assigned to a link-edited module is the first byte in the first control section that is linkage-edited if no main program is included. If a main program is included, the standard entry point is the first byte in the first control section of the main program. This procedure imposes the following rules when linkage-editing FORTRAN modules:

1. If a task requires a BLOCK DATA subprogram, this subprogram must be included prior to any module containing a COMMON block for the same areas defined in the BLOCK DATA subprogram.
2. A linkage-edited module does not retain the module names of those modules included in the linkage editing. Thus the module name in a CALL command must be the name assigned to the linkage-edited

Table 7. STOP/PAUSE/RETURN Differences

FORTRAN STATEMENT	EFFECT IN CONVERSATIONAL MODE	EFFECT IN NONCONVERSATIONAL MODE
PAUSE n or PAUSE 'message'	1. Prints the message "PAUSE n" or "PAUSE message" at the users terminal. 2. Prints an underscore at terminal requesting a command.	PAUSE n or 'message' prints on SYSOUT data set; execution continues with the statement following the PAUSE.
PAUSE n or PAUSE 'message' (cont)	3. Program may be continued at the statement following the PAUSE by entering the RUN command.	PAUSE n or 'message' prints on SYSOUT data set; execution continues with the statement following the PAUSE.
STOP n or RETURN (in main program)	1. Prints "STOP n" at the user's terminal. 2. Prints an underscore at terminal requesting a command.	1. Prints STOP n on the SYSOUT data set. 2. Reads the next command from the SYSIN data set.

module; execution will begin at the standard entry point of the module.

Use of CALL Command and CALL Statement with FORTRAN Subprogram Module Names

It is not good practice to use the CALL command to run FORTRAN subprograms or linkage-edited subprograms without a main program. If this is attempted, some interruption-handling mechanisms will be missing, and the subprogram's execution may be terminated abnormally. A FORTRAN program must not attempt to call a subprogram by its module name.

Initial Content of FORTRAN Variables

FORTRAN programmers should never assume that the value of any variable is known unless the variable has been set by an assignment statement or by a Type or DATA statement in which initial values are specified.

Appendix D. Assembler Language Subprograms

A FORTRAN program can call assembler-language subprograms; similarly, an assembler language program can call a FORTRAN subprogram. This appendix discusses both types of calls. The reader is also referred to *Assembler Language, GC28-2000*; *Assembler User Macro Instructions, GC28-2004*; and *IBM FORTRAN IV Library Subprogram, GC28-2026*. The reader of this appendix must be familiar with the assembler language.

This appendix is divided into the following sections:

1. *FORTRAN Object Program Structure*: general characteristics of FORTRAN-compiled object programs.
2. *Subprogram References*: register use, save areas, and related information.
3. *Types of FORTRAN Calls*: a general description of the manner in which compiler programs call-by-name, call-by-value, and pass subprogram names in a parameter list.
4. *Linkage between FORTRAN and Assembler Language Programs*: Detailed examples for FORTRAN programs calling assembler language subprograms and vice versa.

FORTRAN Object Program Structure

There are certain conditions of which assembler-language users writing code to link to or from FORTRAN object code should be aware:

1. FORTRAN object code is reenterable.
2. A single CSECT and an associated PSECT are generated for each compiled program not containing COMMON.
3. Standard type-I linkages (described below) are compiled for subprogram references.
4. FORTRAN main programs, but not subprograms, issue a call on the CHCDB1 entry to module CHCBD prior to any other operations. This call causes the following (see Appendix C for a more complete discussion):
 - a. Bits are set in the PSW such that fixed-point overflow and significance exceptions are ignored.
 - b. Initialization is performed such that four types of exceptions cause control to be passed to the system modules CHCBD or CHCBE, at which point a flag is set for later interrogation. The four exceptions and the FORTRAN statements that test the flag are:

Exponent-overflow	CALL OVERFL
Exponent-underflow	CALL OVERFL

Floating-point divide Specification	CALL DVCHK See below
-------------------------------------	-------------------------

A specification exception occurs where a variable is not on a proper word boundary. This condition can occur in a FORTRAN program when a misalignment is forced through the use of the COMMON and/or EQUIVALENCE statements. The compiler issues a warning diagnostic if the user has caused such a misalignment, but program execution will be permitted. When a specification exception occurs, an entry in FORTRAN IV-supplied subprogram CHCBE is entered. According to the option chosen by the installation when the system was generated, CHCBE either terminates the program or causes the requested operation to be performed as if no exception has occurred.

- c. Clears any pointers to entries in the DCB table. If an assembler-language main program calls a FORTRAN subprogram and arithmetic overflow and boundary alignment problems are to be handled, the user must either call CHCBD1 as a part of his initialization procedures in his main program or provide assembler-language coding to accomplish this end. CHCBD1 issues SIR, DIR and SPEC macro instructions with the default priority.

Subprogram References

This section gives general information concerning subprogram references by both FORTRAN and assembler-language programs. The following section gives specific examples for both types of references.

The FORTRAN program can refer to a subprogram in two ways: by a CALL statement or by a function reference within an arithmetic expression. For example, the statements

```
CALL SUBR(X, Y, Z)
I=J+K+FUNC(L, M, N)
```

refer to a subroutine subprogram SUBR and a function subprogram FUNC, respectively.

For every subprogram reference, the compiler generates a type-I linkage. Similarly, FORTRAN subprograms expect to be called by a type-I linkage.

Associated with type-I linkage conventions are three areas of concern; these are:

1. Register usage
2. Parameter lists
3. Save areas

Proper Register Usage

TSS/360 has assigned roles to certain registers used in generating a linkage. The function of each linkage register is illustrated in Table 8. Note that registers 2 through 12 are not assigned and, thus, are always available to the user for other purposes.

It is the responsibility of the *called* module to maintain the integrity of general registers 2 through 12 so that their contents are the same at exit as they were at entry to the called program. It is the *calling* program's responsibility to maintain the floating-point registers and program mask around a call. General registers 0, 1, and 13 through 15 must conform to the indicated conventions; 0 and 1 may be destroyed by the called module.

Table 8. Linkage Registers

GENERAL REGISTER	USAGE
0	Integer Result Register (FORTRAN-supplied subprograms).
1	Parameter List Register — contains the address of a list of pointers to input parameters.
13	Save Area Register — contains the address of the calling module's save area.
14	Return Register — contains address in calling module at which execution resumes upon return.
15	Entry Point Register — contains address of the entry point in the called module; also Return Code Register — contains return code set by called module.

By convention, general register 0 is used by a FORTRAN FUNCTION statement subprogram to return the resultant value computed in the subprogram when the resultant value is an integer. When the resultant value is a floating-point number, floating-point register 0 is used. Complex numbers are returned in floating-point registers 0 (real part) and 2 (imaginary part).

Reserving a Parameter Area

If a called module requires input parameters, the calling module must supply the called module with the location of a parameter list in general register 1. Each entry in the parameter list must be on a fullword boundary and represents the address of a parameter being passed to the called module. If the parameter list is variable in length, the length is specified as a count of the number of addresses that compose the list. This count is located one word before the first word in the parameter list. Regardless of whether the parameter list is of fixed or variable length, the parameter list register points to the first word of the parameter list. The CALL macro instruction can be used to generate the parameter list, as well as to link to the called module. The FORTRAN CALL statement does not generate a variable length parameter list count.

Reserving a Save Area

It is the responsibility of the calling module to supply a 19-word area to be used by the called module. Figure 18 shows the layout of the save area and briefly describes the information saved in the area by the calling and called module. Of particular interest in this save area (for trace purposes) are the following two words:

Word 2 — The “backward pointer.” This word always points to the save area of the module that called the module whose save area is being inspected.

Word 3 — The “forward pointer.” This word contains the address of the save area of the module last called by the program whose save area is being inspected. The low-order bit of this word is set to zero as the called program is entered and set to 1 upon exit if the T option in the RETURN macro instruction is used. (The FORTRAN RETURN statement also causes this bit to be set). This bit is useful in determining the flow of control during program execution.

SAREA → (word 1)	Contains the length of the save area in bytes, a minimum of 76.
SAREA + 4 → (word 2)	The address of the calling module's save area. This field is set by the called module in its own save area.
SAREA + 8 → (word 3)	The address of the next save area; that is, the save area of the called module. This field is set by the called module.
SAREA + 12 → (word 4)	The contents of register 14 containing the address to which return from the called module is made. This field is set by the called module in the calling module's save area.
SAREA + 16 → (word 5)	The contents of register 15, containing the address to which entry into the called module is made. This field is set by the called module in the calling module's save area.
SAREA + 20 → (word 6)	The contents of register 0. Value in register 0 set by calling module and saved by called module.
SAREA + 24 → (word 7)	The contents of register 1.
SAREA + 28 → (word 8)	The contents of register 2.
SAREA + 32 → (word 9)	The contents of register 3.
SAREA + 68 → (word 18)	Eight words containing the contents of registers 4-11.
SAREA + 72 → (word 19)	The contents of register 12.
	The address of the PSECT of the called module. This field must be set by the calling module, by storing the R-value of the called entry point in it.

Figure 18. Save Area Format and Content

Variable-Length Parameter Lists

FORTRAN will not provide a TSS/360 variable-length parameter list (except to the DUMP/PDUMP programs),

nor can any TSS/360-supplied subprograms except DUMP and PDUMP process a variable parameter list passed to them.

Types of FORTRAN Calls

A FORTRAN program can call a subprogram with two basic types of parameters.

The first type is one in which a parameter is the name of a variable, the value of which is to be operated on in assignment statements, transfer of control statements, etc. (This is the usual use of parameters in a subprogram reference.) An example of this use follows:

```
C PROGRAM MAIN
  CALL SUBR(A, B)
  .
  .
  SUBROUTINE SUBR(X, Y)
  .
  .
  END
```

The call of SUBR in program MAIN above results in MAIN passing to SUBR in a parameter list the addresses of A and B; both A and B, in this example, lie within MAIN. SUBR references to its dummy arguments X and Y will result in references to A and B within MAIN, as SUBR uses the parameter list passed at the CALL to obtain the location of X and Y.

The above type of calls is referred to as "call-by-name." Some compilers also have implemented "call-by-value." Although the TSS/360 compiler treats this type of call identically with a call-by-name, a brief discussion of call-by-value is given here to note the differences.

In a call-by-value linkage, references to X and Y are treated differently. Rather than referring to the values of X and Y stored in MAIN (and known to MAIN as A and B), a call-by-value causes SUBR to obtain the current values of A and B from MAIN and store them within SUBR. All references to X and Y in SUBR will then refer to the locally stored (i.e., within SUBR) values, rather than the values in MAIN. The values are identical, of course, so the end result is the same.

The second basic type of call is one in which the passed parameter is the name of a subprogram. An example of such a linkage is:

```
C PROGRAM MAIN
  .
  .
  EXTERNAL FUNC
  .
  .
```

```
CALL SUBR(A, FUNC)
.
.
END
SUBROUTINE SUBR(X, BETA)
.
.
Y=BETA(U, V, W)
.
.
RETURN
END
```

In this example, program MAIN does not pass the location of FUNC, as FUNC is not a variable name but is the name of a function type subprogram. In this case, MAIN passes the location of a parameter list, containing the V and R-values for FUNC (rather than the location of a variable, as is the normal case), and SUBR uses this information when referring to the function that SUBR knows as BETA.

Linkage Between FORTRAN and Assembler Language Programs

This section describes the linkage between calling and called programs for calls in which the name of a variable and the name of a subprogram are arguments of the call.

CALL Where the Argument is a Variable Name

In the following example, showing the FORTRAN-assembler language interface, a FORTRAN main program MAIN calls an assembler-language subroutine, ASUBR, which calls a FORTRAN subroutine named FSUBR.

The source statements for the FORTRAN main program are given below, where the leftmost numbers are the system-assigned line numbers, used for reference.

```
0000100C MAIN PROGRAM FOR INTERFACE EXAMPLE
.
.
0000700 CALL ASUBR (A, B)
0000800 STOP 'END OF EXAMPLE'
0000900 END
```

The CALL statement at line 700 generates object code equivalent to the following:

LA	13, MAIN#P	Caller's PSECT and save area base register
LA	15, ASUBVR	V-and R-values for ASUBR
LA	1, PLIST	Parameter list
L	14, 4(15)	R-value
ST	14, 72(13)	To PSECT 19th word
L	15, 0(15)	V-value for ASUBR
BASR	14, 15	Call ASUBR
.	.	.
.	.	.
.	.	.

```

MAIN#P PSECT
      DC   F'76'
      DC   18F'0'
ASUBVR DC   V(ASUBR)
      DC   R(ASUBR)
PLIST  DC   A(A)
      DC   A(B)
      .
      .
      END

```

The purpose of subroutine ASUBR is to exchange the values of A and B.

```

ASUBRP PSECT
      ENTRY ASUBR
      DC   F'76'
      DC   18F'0'
ASUBRC CSECT
      USING ASUBR, 15
ASUBR  SAVE (14, 12)
      L    14, 72(13)
      ST   14, 8(13)
      ST   13, 4(14)
      LR   13, 14
      USING ASUBRP, 13
      LR   12, 15
      DROP 15
      USING ASUBR, 12
      LM   6, 7, 0(1)
      L    2, 0(6)
      L    3, 0(7)
      ST   2, 0(7)
      ST   3, 0(6)
      .
      .
      .
      ( Call
      FSUBR,
      shown
      later )
      .
      .
      L    13, 4(13)
      RETURN (14, 12)
      END

```

The call by assembler-language program ASUBR on FORTRAN subroutine FSUBR is described next.

```

ASUBRP PSECT
      .
      .
      .

```

```

FSUBRVR ADCON IMPLICIT,EP=FSUBR
FSAB    DS     F
      DS     F
ASUBRC  CSECT
      .
      .
      LM     6, 7, 0(1)
      STM    6, 7, FSAB
      .
      .
      LA     15, FSUBRVR
      CALL   (15), MF=(E, FSAB)

```

Many forms of the CALL macro instruction can be used. In the form of the CALL macro instruction used above, the generated code is:

```

      LA     1, FSAB
      L     14, 16(15)
      ST    14, 72(13)
      L     15, 12(15)
      BASR  14, 15

```

The above CALL of the FORTRAN subroutine FSUBR uses a V-and-R-value pair in the linkage. This type of linkage is required, as FSUBR (like all FORTRAN-compiled programs) is reenterable; thus, the called program must include a PSECT. Note also that a program calling a FORTRAN subprogram must provide the address of a 19-word save area in register 13, and the FORTRAN subprogram called modifies the contents of this save area by storing registers in it. Generally, the first 19 words of the caller's PSECT are reserved for this save area, but it is legitimate to point to a save area elsewhere in the program.

CALL Linkage Where the Argument Is a Subprogram Name

An example was given earlier of a main program passing the name of a function FUNC to subroutine SUBR. The FORTRAN statements were:

```

C PROGRAM MAIN
      .
      .
      .
      EXTERNAL FUNC
      .
      .
      CALL SUBR (A, FUNC)
      .
      .
      .
      END
      SUBROUTINE SUBR (X, BETA)
      .
      .
      .

```



```

Y=BETA (U, V, W)
.
.
RETURN
END

```

The code generated in MAIN for this example is the equivalent of the following:

(Program MAIN CSECT)

```

.
.
.

```

```

                L          15, SUBRVR
                LA          1, PARAM
                CALL        (15)
(Program MAIN PSECT)
.
.
.

```

```

SUBRVR  ADCON      IMPLICIT,EP=SUBR
PARAM   DC         A(A)
        DC         A(VRPAIR)
.
.
VRPAIR  DC         V(FUNC)
        DC         R(FUNC)

```

When referring to FUNC (known to SUBR as function BETA), subprogram SUBR does not generate for BETA, but uses the V- and R-values passed in the parameter list.

Using Data in COMMON

Both named and blank COMMONS in a FORTRAN IV program can be referred to by an assembler-language subprogram. To refer to named COMMON, the V-type address constant

name DC V(name of COMMON)

is used.

A blank COMMON may also be defined (by the COM instruction) in an assembler-language program. After the first program containing a blank COMMON is loaded, all blank COMMONS in subsequently loaded programs are rejected; any references to the blank COMMON are tied to the already loaded one, thus making data in the first blank COMMON accessible to more than one program.

Referring to Variables in an Array

For an array, the address of the first variable in the array is placed in the parameter list. An array (for example, a three-dimensional array C(3,2,2)) appears in this format in main storage:

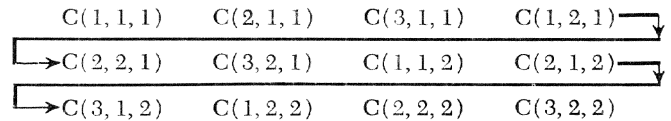


Table 9 shows the general subscript format for arrays of 1, 2, and 3 dimensions.

Table 9. Dimension and Subscript Format

ARRAY A	SUBSCRIPT FORMAT
A(D1)	A(S1)
A(D1, D2)	A(S1, S2)
A(D1, D2, D3)	A(S1, S2, S3)
D1, D2, and D3 are integer constants used in the DIMENSION statement. S1, S2, and S3 are subscripts used with subscripted variables.	

The address of the first variable in the array is placed in the list. To retrieve any other variables in the array, the displacement of the variable (that is, the distance of a variable from the first variable in the array) must be calculated. The formulas for computing the displacement (DISPLC) of a variable for one-, two-, and three-dimensional arrays are

$$\begin{aligned}
 \text{DISPLC} &= (S1-1) \cdot L \\
 \text{DISPLC} &= (S1-1) \cdot L + (S2-1) \cdot D1 \cdot L \\
 \text{DISPLC} &= (S1-1) \cdot L + (S2-1) \cdot D1 \cdot L + (S3-1) \cdot D2 \cdot D1 \cdot L
 \end{aligned}$$

where

L is the length of each variable in the array.

For example, the variable C(2,1,2) in the main program is to be moved to a location ARVAR in the subprogram. Using the formula for displacement of variables in a three-dimensional array, the displacement (DISP) is calculated to be 28. The following instructions can be used to move the variable:

```

LA      6, 8(13)
LA      8, DISP
L       9, 0(6, 8)
ST      9, ARVAR

```

Appendix E. Specification of Data Set Characteristics

This appendix discusses eight topics associated with defining and processing data sets with FORTRAN object programs. The first of these, "Data Set Creation and Structure," gives a description of the format, creation, and structure of data sets written and read by FORTRAN-compiled modules.

Next, the section "Operation on Data Sets," describes techniques for: creating new data sets, print and punch output, processing data card input, and reading data sets created by other TSS/360 FORTRAN programs or OS/360 FORTRAN programs. Also included are discussions of: exception handling (I/O errors and end-of-data-set handling); use of REWIND, ENDFILE, and BACKSPACE statements; and execution I/O error messages.

The next section, "SECURE Requirements for Non-conversational Tasks," includes considerations in determining the number of private devices needed during processing of a nonconventional task.

"Guide to DDEF Commands" describes in detail how to write DDEF commands for all allowable FORTRAN object-time processing. This section is divided into a description of the basic DDEF command, which may satisfy all normal user requirements; the default of DDEF commands, which discusses SYSIN/SYSOUT I/O; and the full DDEF command, describing extended processing facilities.

The section "Sample DDEF Commands" presents and explains a variety of DDEF command uses. The final three sections of this appendix are "Error Messages for the DDEF Command," "Data Definition Rules for Language Processing," and "Data Definition Rules for TSS/360 Commands."

Data Set Creation and Structure

This section describes the format, creation, and structure of data sets written and read by FORTRAN-compiled modules.

In creating and using data sets, the user is concerned with two things: the logical record format and the data set organization. A *logical record* is the unit of information processed by the user's program — that is, the specific number of contiguous bytes of information that is to be read or written by a FORTRAN program. TSS/360 recognizes three formats for logical records:

1. Format F, for logical records of a fixed length.
2. Format V, for logical records of varying length.
3. Format U, for logical records of undefined format.
4. Format D, for ASCII tape records.

Detailed descriptions of these three formats are given later in this section.

The data set organizations are spoken of in terms of the TSS/360 *access methods* used to manipulate them; that is, the particular set of system routines that are used to transfer data records between virtual storage and a data set in external storage that is organized in a particular way. The TSS/360 access methods are described below.

Access Methods

Data set records are transferred to and from I/O devices and virtual storage by system programs known as access methods. There are two primary access methods.

1. *Virtual Access Method (VAM)*: The access method used in TSS/360 unless the data sets must be interchanged with programs running in the IBM System/360 Operating System or the Model 44 Programming System, or the data set is to be written on magnetic tape.
2. *Basic Sequential Access Method (BSAM) and Queued Sequential Access Method (QSAM)*: Used to read and write records that can be read and written with programs running under control of the IBM System/360 Operating System or the Model 44 Programming System, or when the data set is to be written on magnetic tape.

The choice of access methods and record formats (see below) are originally determined by the parameters in the DDEF command. A later section of this appendix contains a complete discussion of how DDEF commands are written.

Virtual Access Method

Users create, read, and process virtual access method (VAM) data sets on the basis of logical records. The system, however, blocks these records by pages (4096 bytes) and uses the page as the unit of transfer between the direct-access device and the user's virtual storage. The system also ensures that only those pages of a data set that are actually required are resident in virtual storage.

Virtual storage data sets can be classified as:

1. Virtual sequential (vs)
2. Virtual index sequential (vi)
3. Virtual partitioned (vp)

VAM data sets *must* reside on direct-access volumes (not tape) that are specifically formatted for TSS/360. Labels and other related tables pertaining to the data set for VAM are not the same as those in an OS/360-generated direct-access data set. Therefore, data sets written with VAM cannot be read with a program running under the IBM System/360 Operating System or the Model 44 Programming System.

For FORTRAN, format-U records in any VAM data set are created as format-F with a record length of 4096 bytes.

Virtual Sequential (VS): In a virtual sequential (vs) data set, the order of the logical records is determined by the order in which they are created. vs permits logical records up to 1,048,576 bytes (256 pages) in length.

Most FORTRAN programs read and write logical records using vs in format V. This access method, with this format, is sufficient to handle nearly every application, and only when records are specially formatted will the user have to use some other combination of access method and record format.

Virtual Index Sequential (VI): Virtual index sequential records are similar to virtual sequential records with the addition of an extra field called the key.

In vi data sets, it is only possible to have format-V or -F records; format-U records are not permitted. Logical record length is limited to 4000 bytes; key length cannot exceed 255 bytes, but keys can be anywhere in the record.

The records in the data set are ordered by ascending sequence of the key field. A FORTRAN program can read any vi data set sequentially and records are presented in ascending key order. FORTRAN cannot read vi records by key — that is, nonsequentially. When reading a vi record, the user must account for the key in his FORMAT statement since the key is considered part of the record and must be maintained by the user. An assembler-language subprogram may be used to read records nonsequentially. Writing a vi data set with a FORTRAN program can be done sequentially; however, the key field in the record is checked and used by data management. Therefore, the user must lay out the logical records so that an ascending key, according to the System/360 8-bit code, appears in the same place on every record. For example, the following FORTRAN statements could be used to write a vi data set which contains a 6-character ascending key at the beginning of each logical record.

```

DIMENSION DATA (40)
KEY = 0
.
.
10 KEY = KEY + 1
WRITE (7, 200) KEY, DATA
200 FORMAT (I6, 40F5. 1)
.
.
GO TO 10

```

To summarize, since TSS/360 FORTRAN does not have random access capability, vi records are of limited use to the FORTRAN user. vi would be used primarily to construct records that are to be read by non-FORTRAN programs, and in writing these records it is the responsibility of the user to supply the KEY in the record.

Virtual Partitioned (VP): Partitioned data sets may contain both vi or vs data set organization. Logical records are grouped into named subdivisions so that processing can take place on any one of them, called a *member* of the partitioned data set. Each member can be treated as an individual data set.

To create and operate on a member of a partitioned data set, the user writes a DDEF command to (1) indicate a DSORG of either VSP or VIP in the subfield of the DCB parameter and (2) append a member name (in parentheses) to the DSNAME. Only one DDEF command can be issued for the partitioned data set at one time; hence, only one member may be processed at any one time.

Physical Sequential (PS)

Physical Sequential (ps) data sets are written by the basic sequential access method (BSAM), and queued sequential access method (QSAM). However, this is normally done only to communicate with the IBM System/360 Operating System or the Model 44 Programming System, or if the data set is to be written on magnetic tape.

BSAM can support any of the record formats — F (blocked and unblocked), V (blocked and unblocked), U (unblocked only), and D (blocked and unblocked). If the user wants to process blocked records with BSAM, he must perform all deblocking first. If the user needs to process blocked records, FORTRAN I/O will automatically perform the deblocking function.

QSAM, on the other hand, supports both blocked and unblocked records. In the case of blocked records, QSAM will automatically provide the deblocking function of the user.

ps data sets are built around physical blocks that contain one or more logical records in format F, V, or U. Format-U records are considered unblocked.

ps data sets may be written either on magnetic tape or direct-access devices. To read a ps data set the user

must first ensure that it is available to the system on the correct device and then indicate to the system the data set's special characteristics. These two functions are performed by special parameters in the DDEF command. (These are described in a later section of this appendix.)

Data Set Records

A logical record is a specified number of contiguous bytes of information that is to be read in or written out by a program.

1. *Format F*: If a data set is made up of records that are all of the same length, it is format F, for fixed length. There are no special restrictions on the contents of a format F logical record; however, different access methods set limits on record size (see Figure 18).
2. *Format V*: If a data set is made up of records that are of varying length, it is format V, for variable length. Again, the different access methods set limits on record size.
3. *Format U*: A third class of record is format U, for undefined length. For records on vs data sets, format U records are always considered to be format F with a record length of 4096 bytes or a multiple of 4096; for records on ps data sets, the record length is determined by some physical boundary that is recognized by the device that reads the record.
4. *Format D*: If a data set contains ASCII records, format D must be specified. Format-D records are variable in length.

The above considerations for using format-F records with FORTRAN programs do not apply to unformatted READ or WRITE records.

Record Format	VS	VI	PS	
			Unblocked	Blocked
F	1,048,576	4000	32760	32760
V	1,048,572	4000	32756	32756
U	4096*		32760	
D			32760	32760

*All U-format records are multiples of 4096.

Figure 19. Maximum Record Lengths (Bytes)

Variable-Length Format

1. *Record Lengths*: vs format-V records can contain from 1 to 1,048,572 bytes of data. vi format-V records can contain from 1 to 3096 bytes of data. ps format-V records can contain from 1 to 32,752 bytes of data.

The record length is indicated at the start of each record. ps records (blocked and unblocked) have a two-byte binary number followed by two bytes

reserved for system use. vi and vs blocked records have a four-byte binary number. FORTRAN-written programs can read ps data sets and write them on vi or vs and vice versa.

2. *Reading and Writing*: FORTRAN I/O processes input and output format-V records exactly as if they were format F (i.e., the record-length bytes are not transmitted into user storage nor do they need to be constructed by the user before issuing a WRITE). No padding of output records ever takes place. All unformatted FORTRAN records are written as format V with spanning control bits in either the first byte of a vi or vs record or the third byte of a ps record; again this special byte is handled by the system.

Both format-V and -F records can occur in any of the different data set organizations, although they cannot be mixed in any one data set. The organization to be used for an output data set is specified in the DDEF command. The organization used for an input data set can be determined from the data set standard labels, from information in the system catalog, or from the DDEF command.

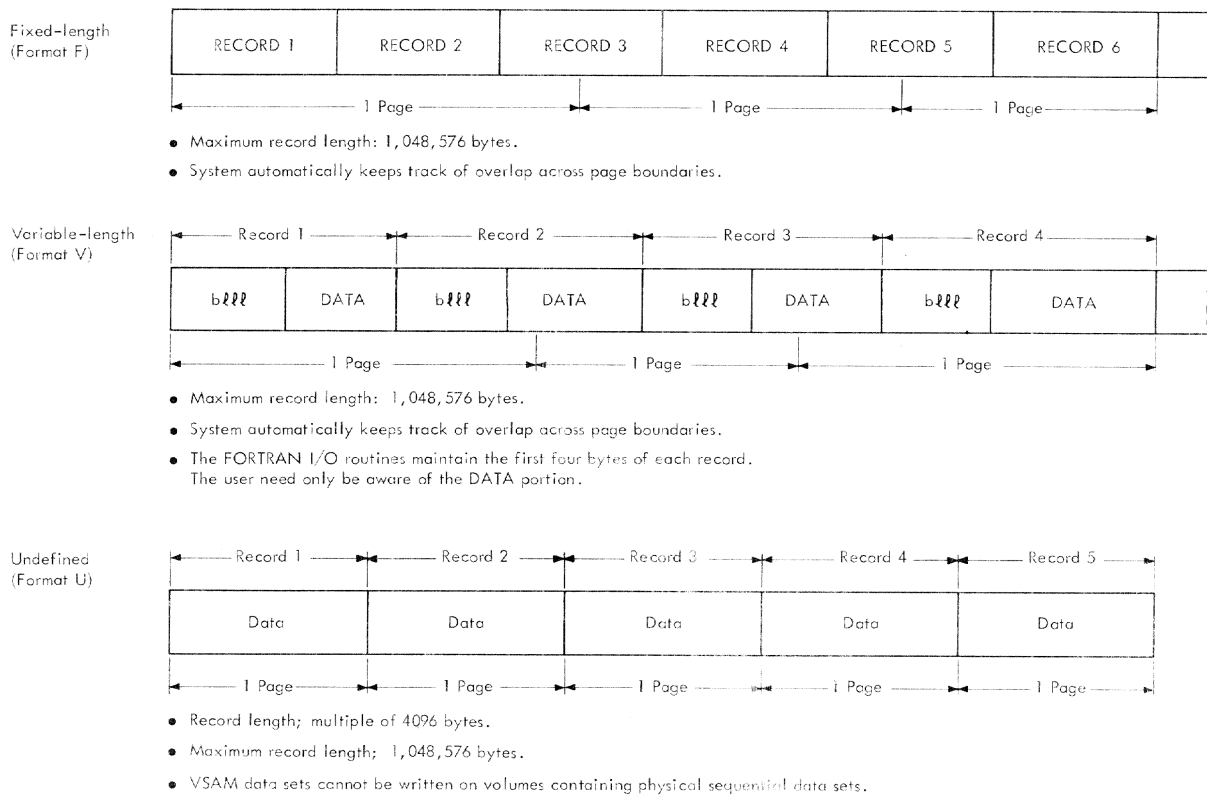
FORTRAN I/O always writes format V records unless otherwise instructed by the DDEF command.

The user's source program need never be concerned whether format-V or -F records are being read or written, since FORTRAN will construct the records properly and transmit them to the I/O device as instructed. The same is true for format-U records, but their use should be restricted to ps data sets, since VAM generates very large records for format U.

Figures 20, 21, and 22 show the various records formats under different data set organizations.

Fixed-Length Format

1. *Record Lengths*: vs format-F records can be any length from 1 to 1,048,576 bytes. vi format-F records can be any length from 1 to 4,000 bytes. ps format-F records can be any length from 1 to 32,760 bytes. Unblocked records less than 13 bytes in length are rejected if a reading error occurs (i.e., no recovery attempt is made). Therefore, unblocked records of less than 13 bytes are not recommended. Blocked records are automatically at least 13 bytes long.
2. *Blocking format-F records*: Any specification of block size (in the DDEF command) is ignored for blocking of format-F records in VAM data sets. The block size of format-F records in ps data sets must be an integral multiple of the record size.
3. *Writing*: If a data set is defined as containing format-F records, any short records written by a FOR-



Note: Internal record formats may differ from the external record formats described in this book.
 For an explanation of internal record formats, see IBM System/360 Time Sharing System: Access Methods PLM, GC28-2016.

Figure 20. Record Formats – Virtual Sequential

TRAN program are padded with trailing blanks or binary zeros to the required size. Attempts to write logical records greater than the size specified will cause diagnostics and the record will be split.

4. **Reading:** Unused portions of an input record cause no error indication and are simply ignored. Attempts to read logical records longer than the size of the format-F records causes a diagnostic and the next logical record will be read to fulfill the request.

FORTRAN Records

Within the four classes of logical records (i.e., F, V, U or D), the FORTRAN I/O routines construct three different types of records, depending on the FORTRAN language statements used to write the records. These three types of records are formatted, NAMELIST, and unformatted.

Formatted Records

A FORTRAN program most commonly reads or writes logical records whose length is defined by the FORMAT statement and the list. Thus:

```
FORMAT (15H RESULTS ARE )
```

can be referred to in a WRITE statement without list in

order to write a single logical record of 15 bytes. Alternatively:

```
FORMAT (15H RESULTS ARE , 5I5)
```

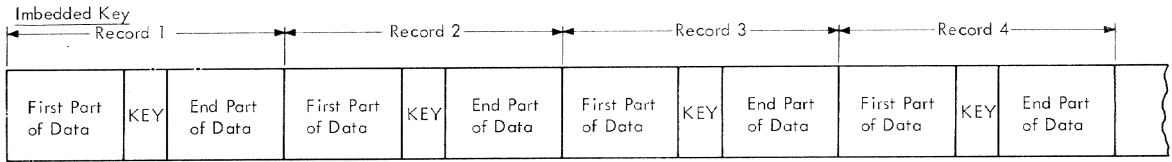
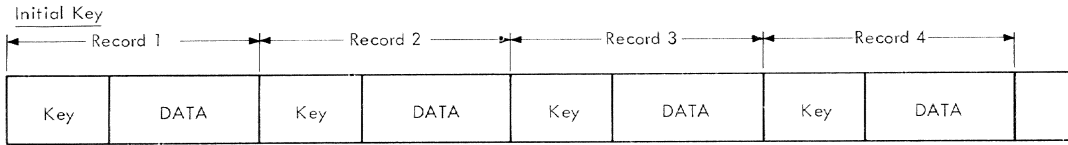
produces a logical record of 20 bytes if the WRITE statement has a single (simple) variable name in the list, 25 bytes for two names, etc., up to 40 bytes for five names in the list. If there are more than five simple variables in the list, a number of additional records are written, all of which (except possibly the last) are 40 bytes in length, since the FORMAT specification specifies 40-byte records.

When FORTRAN I/O has built up a logical output record according to the rules specified by the FORTRAN IV language, it incorporates the resulting record into any data set format, adding a length indicator, if required for format-V records, and making up a block for F format blocked records.

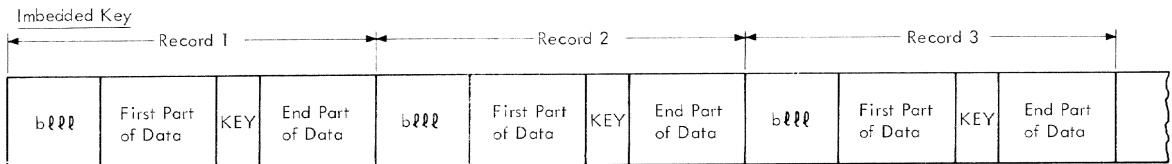
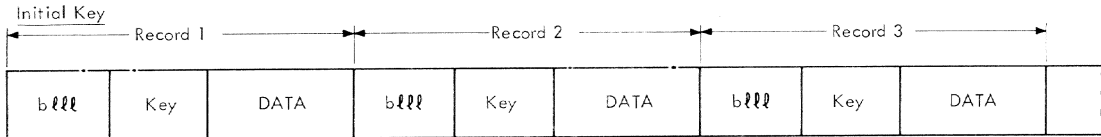
If the user has requested format-F or -V records for an output data set (in a DDEF command), none of the generated logical records should exceed the maximum permitted logical record length specified in the DDEF. Two or more data set records are generated in such a case, and a diagnostic is issued.

If an attempt is made to read a logical record longer than the one on the input data set, a diagnostic mes-

Fixed-length
(Format F)



Variable-length
(Format V)



- Maximum logical record length: 4000 bytes.
- Maximum number of records per data page: 1300.
- Maximum key length: 255 bytes.
- Maximum number of data pages: 65,000.
- Maximum number of overflow pages: 240.
- Maximum number of records per overflow page: 255.
- No limit to the number of directory pages.
- User responsible for Key and DATA parts of each record.
- FORTRAN I/O maintains first four bytes of format-V records.

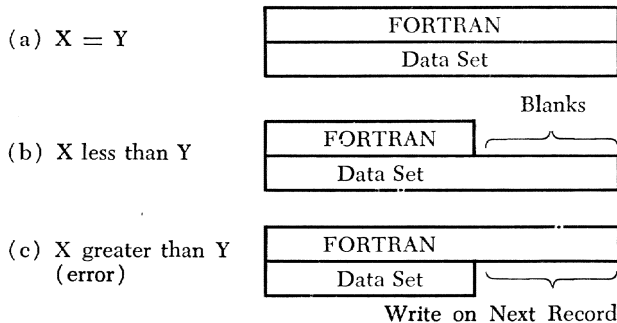
Figure 21. Record Formats – Virtual Index Sequential

sage is issued and the next record will be read to fulfill the READ.

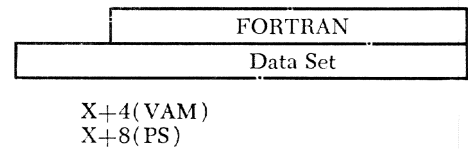
Parameters in the `DDDEF` command that differ from those of an input data set override the latter, and will cause errors.

A summary of formatted FORTRAN records is as follows:

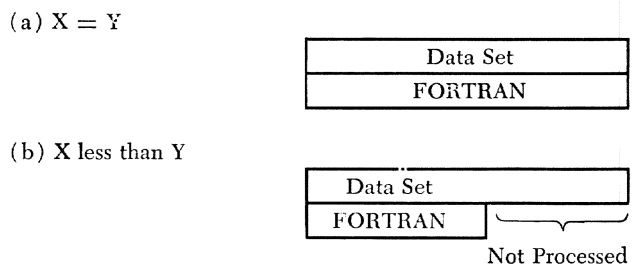
1. FORTRAN logical record of X bytes written on a data set with format-F records with a logical record length (LRECL) of Y bytes:



2. FORTRAN logical record of X bytes written on a data set with format-V records. LRECL is unimportant, provided it is not set to less than X.



3. FORTRAN logical record of X bytes read from a data set with format F records with an LRECL of Y bytes:



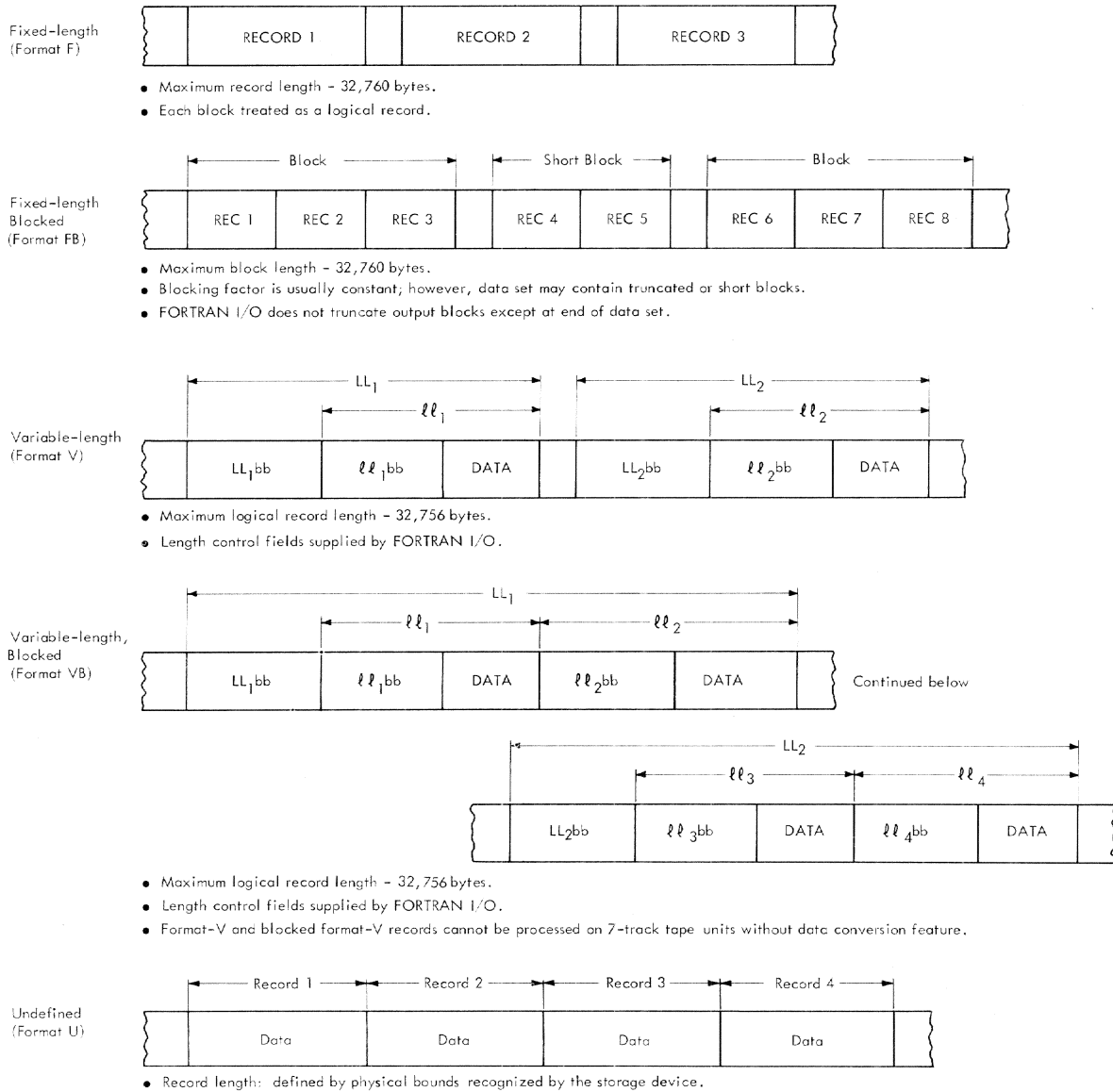
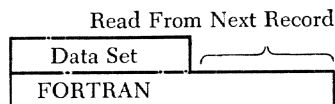
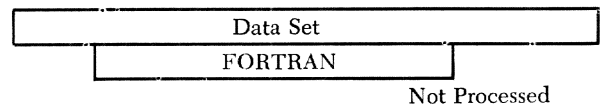


Figure 22. Record Formats - Physical Sequential

(c) X greater than Y (error)

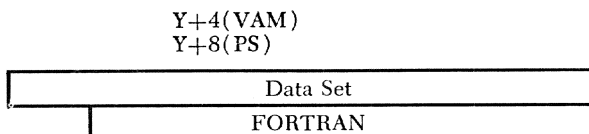


(b) X less than Y

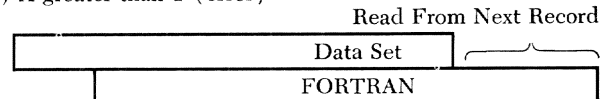


4. FORTRAN logical record of X bytes read from a format V data set logical record with a data length of Y (data set LRECL in the DDEF command is unimportant unless it is deliberately set smaller than maximum likely Y):

(a) X = Y



(c) X greater than Y (error)



NAMELIST Records

FORTRAN NAMELIST records are written based on a NAMELIST variable list that provides a NAMELIST designator name followed by a number of variable or array names.

This results in the generation of at least three logical records:

1. A record containing only the `NAMELIST` designator name preceded by an ampersand (&), which starts in character 2 of the record
2. One or more records containing items of the form `variable-name = integer, real, complex, or logical number`
3. A final record containing `&END`, which starts in the second character of the record.

The user has no control over the spacing or size of items in the second class of record; this depends on the size of the number field required to represent the variable. Typical output might appear as:

```
&NAME1
A=47.00000,I=5,N=-76709,XXX=-0.1234567E-07
&END
```

If format `V` is specified, records of different lengths are written, but none exceeds 120 bytes.

If format `F` is specified, the specified logical record length in the `DDEF` is used. The logical record length can be set to 120 to facilitate subsequent printing of the data set.

If format `U` is specified (with a data set organization of `PS`), unblocked records (format `V`) are directed to the output device.

If the `DDEF` command for a `NAMELIST` output data set is defaulted, output is directed to `SYSOUP` as variable-length records of up to 120 bytes. It is either written conversationally at the user's terminal or stored on a system data set (`SYSOUP`) for subsequent printing.

Since key information cannot be maintained within `NAMELIST` records, `NAMELIST` records with a data set organization of `VI` are not permitted.

If `NAMELIST` input is from `SYSDIN`, the terminal entries for conversational input should be exactly the same as would be punched on a data card. A prompting message requesting `NAMELIST` input is printed.

Unformatted Records

Unformatted FORTRAN logical records are written only under the control of a list with no `FORMAT` statement. Data is transferred to the output data set in internal representation copied from virtual storage.

All unformatted output is made up into format-`V` records using, if necessary, a standard pair of bits indicating that a given record is incomplete and extends into the next record. For `VSAM` records, the first two bits of the first byte in the record length field are used. For `BSAM` records, the last two bits of the third byte in the block length field are used. The bits of this byte are given the following meaning:

	VSAM	BSAM	Meaning
Bit 0 (first)	6		This record spans into the next record.
Bit 1		7 (last)	This record spans from the last record.

Unformatted records written by the IBM System/360 Operating System with `PS` organization are processed correctly by `TSS/360`.

When the `DDEF` command for an output data set is defaulted and unformatted records are written, they are converted to hexadecimal and appear in this form at the terminal or in the listing. The spanning indicator and record length are printed. When the `DDEF` command for an input data set is defaulted and unformatted records are to be read, the prompting message reminds the user that hexadecimal data should be entered. The user is not required to enter hexadecimal data for the length field and spanning indicator. If one line of input is not enough to fill the elements of the list, additional requests are made for more lines until enough data is received. Invalid hexadecimal characters cause a request for a new line of data. A blank in the input line is treated as marking the end of hexadecimal data, and any further characters in the line are ignored.

Summary of FORTRAN Data Set Formats

Table 10 summarizes the allowable data set organizations and logical record structure for data sets created or processed by FORTRAN object programs.

FORTRAN Operations on Data Sets

Generation of New Data Sets

A FORTRAN program can write an output data set to be used as intermediate data, print or punch output, or to be processed later by other programs.

1. *Intermediate Output:* This requires use of the simplest form of `DDEF` described under basic `DDEF` command. Default characteristics are suitable for all applications and for any combination of `READ`, `WRITE`, and control statements.
2. *Print or Punch Output:* The default parameters of the basic `DDEF` command are acceptable. Unformatted records should not be interspersed with formatted. The `PRINT` or `CATALOG` command should be issued for the data set before `LOGOFF`. In conversational operation, defaulting the `DDEF` command will cause the output to be printed at the terminal.
3. *Output Retention Within TSS/360:* The simplest form of `DDEF` command can be used, and the data set will be cataloged since new data sets defined by the basic `DDEF` command reside on public storage. The data set may reside on either a public or private volume.

Table 10. Data Set Format Summary

DATA SET ORGANIZATION	RECORD FORMAT	LOGICAL RECORD LENGTH (LRECL) NOTES	(DISP=NEW) REQUIRED PARAMETERS IN DDEF	NOTES
VS	F ²	<ol style="list-style-type: none"> 1. Max length = 1,048,576 2. Default length = 133 3. Padding occurs if output less than specified 4. Error if output greater than specified 	<ol style="list-style-type: none"> 1. FTxxFyyy, VS 2. DSNAME= 3. RECFM=F 4. LRECL= 	<ol style="list-style-type: none"> 1. Records are always blocked. 2. Do not use for unformatted I/O. 3. No tapes.
VS	V ¹	<ol style="list-style-type: none"> 1. Max length = 1,048,572 2. Default length = 4092 3. No padding if output less than specified 4. Error if output greater than specified 	<ol style="list-style-type: none"> 1. FTxxFyyy, VS 2. DSNAME= 	<ol style="list-style-type: none"> 1. Records are always blocked. 2. Can be used for <i>formatted</i>, <i>unformatted</i> and <i>NAMELIST</i> logical records. 3. 4-byte length indicator at beginning of each logical record, not transmitted to user's storage. 4. No tapes.
VI	F	<ol style="list-style-type: none"> 1. Max length = 4,000 2. Default length = 133 3. Padding occurs if output less than specified 4. Error if output greater than specified 	<ol style="list-style-type: none"> 1. FTxxFyyy, VI 2. DSNAME= 3. RECFM=F 4. KEYLEN= 5. RKP=³ 	<ol style="list-style-type: none"> 1. Records are always blocked. 2. Do not use for unformatted I/O. 3. User must supply key in the record. 4. No tapes.
VI	V ¹	<ol style="list-style-type: none"> 1. Max length = 4,000 2. Default length = 4,000 3. No padding of logical records 4. Error if output greater than specified 	<ol style="list-style-type: none"> 1. FTxxFyyy, VI 2. DSNAME= 3. KEYLEN= 4. RKP=³ 	<ol style="list-style-type: none"> 1. Records are always blocked. 2. Can be used for <i>formatted</i> logical records (and <i>unformatted</i> using core in the key). 3. User must supply key in the record. 4. 4-byte length indicator at beginning of each logical record, not transmitted to user's storage. 5. No tapes.
PS	F	<ol style="list-style-type: none"> 1. Max length = 32,760 2. Default length = 133 3. Padding occurs if use for output less than specified 4. Error if output greater than specified 	<ol style="list-style-type: none"> 1. FTxxFyyy, PS 2. DSNAME= 3. RECFM=F 4. LRECL= 	<ol style="list-style-type: none"> 1. Records are unblocked. 2. Volume may be tape. 3. Do not use for unformatted I/O. 4. IBM System/360 Operating System volumes can be written/read.
PS	FB	<ol style="list-style-type: none"> 1. Max length = 32,760 2. Default length = 133 3. Padding occurs if output less than specified 4. Error if output greater than specified 	<ol style="list-style-type: none"> 1. FTxxFyyy, PS 2. DSNAME= 3. RECFM=FB 4. LRECL= 5. BLKSIZE= 	<ol style="list-style-type: none"> 1. Maximum block length is 32,760. 2. Default block length is 3990. 3. Block length must be a multiple of LRECL. 4. Can be used for <i>formatted</i> logical records only. 5. Volume may be tape. 6. IBM System/360 Operating System volumes can be written/read. 7. Cannot be specified for ASCII data sets.
PS	V ¹	<ol style="list-style-type: none"> 1. Max length = 32,756 2. Default length = 4092 3. No padding 4. Error if output greater than specified 	<ol style="list-style-type: none"> 1. FTxxFyyy, PS 2. DSNAME= 	<ol style="list-style-type: none"> 1. Maximum block length is 32,760. 2. Default block length is 4096. 3. Block length must be a multiple of LRECL. 4. Can be used for <i>formatted</i>, <i>unformatted</i> and <i>NAMELIST</i> records. 5. Volume may be tape. 6. IBM System/360 Operating System volume can be written/read. 7. 4-byte length indicator at beginning of each logical record, not transmitted to user's storage.

¹IBM-supplied default parameter²Format-U records are treated as Format F with length of 4096 in VS³RKP must allow for the 4-byte indicator

If the data set is to be used subsequently by others, a PERMIT command must be issued for the data set so that those who wish to use it can SHARE it.

4. *Output Retention Outside TSS/360*: A data set to be used on another TSS/360 system is treated as if it were to be used on the same system, but a private volume must be requested. However, if the data set is to be used on another system (not a TSS/360 system), a full DDEF command is required for one of the PS data set types on either direct-access storage or tape. The data set should be on a private volume. All tapes should have standard labels, so that they can be processed on TSS/360 and most other systems) with a minimum of DDEF command parameters.

Reading Existing Data Sets

A FORTRAN object program may use input data sets from a number of different sources and with different characteristics.

From Outside TSS/360

1. *Card decks*. Card decks may be placed in a terminal card reader and read directly by FORTRAN when the DDEF command for the input data set is defaulted. Another method is to submit to the TSS/360 operations center the card deck made up as input to a system program that generates VAM data sets from card decks. The card deck is set up for a nonconversational task in one of two ways:
 - a. The deck contains a LOGON card and command cards that CALL a specified FORTRAN program. The data cards follow the CALL card; the last data card is followed by a card with %END in columns 1-4, by additional system command cards, and finally by a LOGOFF card. The entire deck causes FORTRAN execution similar to that in a batch environment. The resulting VAM data set serves as SYSIN but cannot be cataloged or retained in the system. (See Example 13 in this manual.)
 - b. A data descriptor card immediately precedes the first data card. The data descriptor card is described in *Command System User's Guide*. Following the last data card is a card with %ENDDS starting in column 3. The resultant data set is cataloged and is accessible to FORTRAN programs by means of the basic DDEF command.
2. *Magnetic Tapes*. If the tape is to be used frequently, the system operator should be requested to make a VS data set from the tape using a program similar to that mentioned above for bulk input from cards.

The operator will create a standard cataloged VS data set that is accessible to the user for any session.

The alternative approach is to treat the tape as a private volume, request it on DDEF commands with DSORG set to PS, and specify the additional parameters as discussed in "Full DDEF Command" in this appendix. When the tape has no labels, a number of DDEF command parameters must be specified that are normally obtained from the tape label. However, if the data set is cataloged (CATALOG command following DDEF command), it again becomes possible to use the basic DDEF command.

3. *Direct-Access Volumes (Not VAM)*. To use direct access volumes from outside TSS/360, supply appropriate DDEF commands for DSORG of PS with the parameters given in this Appendix. Issuing a CATALOG command following the DDEF command will make PS-type direct-access volumes more convenient to process in future sessions.

From Other Than FORTRAN Programs on TSS/360

FORTRAN I/O can be used to process most input VAM data sets created under TSS/360. The basic DDEF command is used. Any input logical record that can be represented as a print line can be processed by a FORTRAN program having the right FORMAT statement.

If a non-FORTRAN program has written a non-VAM data set such as a tape or direct-access volume on TSS/360, the user should proceed as if the volume had originated outside TSS/360.

From FORTRAN Programs on TSS/360

There are no special considerations when exchanging data sets between TSS/360 FORTRAN programs.

Exception Handling

If a READ statement is being executed and it contains the optional specification of END= or ERR=, and an end or error condition exists, control transfers to the user program without error message or interruption.

If an end-data-set condition exists and END= causes continued execution, further READS on the same data set cause a search for another DDEF command (e.g., one with a DDNAME of FT01F002 instead of FT01F001).

If an error condition exists, the user can attempt to backspace and read again as many times as he wants.

If DSORG is PS for magnetic tape, the system's error recovery procedures are applied to read or write errors unless suppressed by the IMSK option in the DDEF command. If the error is unrecoverable, the program terminates (system EXIT is called by the I/O routines) unless a READ with the ERR option is being processed. If the IMSK option has been used to sup-

press one or more of the standard error procedures, this suppresses either termination of the program or transfer to the ERR location.

If DSORG is PS for direct access, the system's error recovery procedures attempt to use alternate tracks. Thus, when a track is found to be defective, the system assigns the next available alternate track. The FORTRAN user has no control over this action and is not aware of its taking place except that in the event that no alternate track is available for a WRITE, the program will be terminated.

Positioning Statements and Sequence Rules

When operating with VAM or PS-type data sets, the system attempts to interpret and execute each FORTRAN positioning statement in accordance with the organization of the data set in question.

1. For SYSOUT and SYSIN, all positioning statements are ignored unless they violate rules 2 through 5.
2. An attempt to use ENDFILE on a data set that has so far only been used as input causes an error message and execution is terminated.
3. When ENDFILE is used on an output data set, subsequent WRITE statements relate to a different ddname (i.e., another DDEF command), such as FT01F002 instead of FT01F001.
4. The sequence of statements WRITE-ENDFILE-READ cause an error message and program termination. The sequence WRITE-ENDFILE-REWIND-READ is acceptable.
5. An initial REWIND has no meaning for VAM data sets that are always accessed starting at the first record; hence, it is ignored. The sequence READ-REWIND-READ is acceptable and causes a return to sequential processing starting at the first record of the first data set associated with the data set reference number. Also READ-READ-BACKSPACE-READ is acceptable. Attempts to BACKSPACE when positioned at the first record of a data set are ignored.

Execution I/O Error Messages

All FORTRAN I/O error messages are directed to SYSOUT; that is, they appear at the terminal during conversational mode or are listed after a nonconversational task is complete.

All messages except those that cause a system pause start with a five-character name that identifies the I/O routine issuing the message. The names always start with the four letters CHCI and end in one of the letters A through Y.

Most messages consist of a fixed portion, usually 40 bytes in length, followed by a variable portion of about 20 bytes that contains the variable information associated with the message.

When a FORTRAN PAUSE statement is executed, the user is permitted to enter any command other than LOAD, UNLOAD, or CALL name, and then to continue execution of the program with GO. In nonconversational mode, the PAUSE message is written on SYSOUT and program execution continues.

Certain error conditions do not permit further execution of the program, such as an end-data-set condition from a READ statement with no END= specified. This causes the printing of a message; then control is transferred to the system termination routine, EXIT. Any data sets that the program has left open can be closed with the CLOSE command.

SECURE Requirements for Nonconversational Tasks

Nonconversational tasks are enqueued until the system is able to fill the requirements for private devices. This list of requirements is made available to the system by means of a SECURE command which the user must include in the task's command procedure as the first command after LOGON. Then as each DDEF is read and processed, the required devices are allocated from those that have been secured for the nonconversational task. Any attempt to allocate more than are available will cause the task to be terminated.

In determining the number of devices needed in a task, the following points should be considered:

- The number of devices should be at least equivalent to the number of data sets on different private volumes which are opened at any one time. Two or more data sets residing on the same private volume may require only one device (the exception is described below).
- If two different data sets residing on separate volumes are used in sequence (i.e., the first is closed before the second is opened) the system can be directed to allocate the same device to both by including the UNIT=AFF option in the second DDEF along with the DDNAME of the first DDEF command. When the UNIT=AFF option is selected, the device types of both data sets must be compatible, and neither should be new data sets residing on direct-access devices.
- If two different data sets on the same or different private volumes are defined by the same DDNAME, the UNIT=AFF option may not be selected, regardless of whether the references are in the same module or not. If the same data set reference number is used in different programs in a multiple execution task, then the first data set must be released prior to the second DDEF; thus, two devices must be secured for

the data sets even though both data sets are not open at the same time. If both data sets are processed serially in the same program (i.e., the end of the data set is reached, followed by a READ on the same data set reference number) then two DDEF commands are necessary with the same data set reference number in their DDNAME and with successive data set sequence numbers. In this case, one device is sufficient because the unit affinity option may be used.

If several data sets are to be serially processed with unit affinity specified, each data set may have unit affinity with only the most recently processed data set. Note that unit affinity may only be specified for physical sequential data sets.

If, however, the data sets are not processed serially in the same program, the first data set must be released prior to the second DDEF. Therefore, two devices are necessary since the RELEASE command removed the device from the task's allocation prior to the second DDEF command.

Guide to DDEF Commands

This section discusses:

1. Basic DDEF Commands — describing the general form of the basic DDEF command and its uses.
2. Default of DDEF Commands — describing the conditions where DDEF commands are defaultable and subsequent system action.
3. Full DDEF Command — describing the general form of the full DDEF command for the FORTRAN user. Included is a table illustrating for each type of data set, the required and optional DDEF parameters.

The DDEF command is used to establish a data set in the system and describe its characteristics. In general, any data set required by a FORTRAN object program during execution must be specified in a DDEF command.

A DDEF command can be issued at any time within the session prior to the CALL command for the program in which the data set is to be used. Each DDEF command is valid only during the task in which it is issued; previously defined data sets must be redefined at every task that refers to them. A DDEF command that has been entered can be canceled by a RELEASE command.

Normally, FORTRAN users require only basic DDEF commands, defaulting almost all of the operand fields. In some cases, DDEF commands themselves can be defaulted, in which case the SYSIN or SYSOUT data set for input and output respectively is chosen. More complex

DDEF commands can be used if the data formats require it.

Input/output statements, such as READ, WRITE, REWIND, BACKSPACE, and ENDFILE, apply to collections of data that are referenced within a FORTRAN program as integer numbers:

```
READ (23)A OR WRITE (J)B
```

Since the reference is to the data rather than any specific device, this number is called the *data set reference number*.

The data set reference number used within a FORTRAN program must be associated with a data set name or DSNNAME before the system can read or write it. This relationship is established by reference to a DDEF command that links a data set reference number, a data definition name (DDNAME), and a data set name (DSNAME).

Command format specification conventions are listed in Appendix G.

Basic DDEF Command

The basic DDEF commands may be used for *any* cataloged input data set except those on unlabeled tapes. For a new data set it specifies public volume residence, a virtual sequential (vs) data set organization, and variable length records (format va). Data sets defined with this basic DDEF command must be cataloged by the system. The basic DDEF command is shown in Figure 23.

Operation	Operand
DDEF	DDNAME=data definition name, [DSORG={VI VS VP}], DSNAME=data set name

Figure 23. Basic DDEF Command

DDNAME=FTxxFyyy

The DDNAME parameter must be of the following format:

```
FTxxFyyy
```

where xx is the data set reference number used within the program and must be two integer digits in the range 00-99; for example, a program containing the FORTRAN statement READ (5, 60)A requires a DDEF command with DDNAME parameter FT05F001. The yyy is the three integer digits 001, except as noted below. The yyy portion of the DDNAME permits operations on multiple data sets with the same data set reference number. For example, when a READ (5, END=40) is used and an end-of-data-set condition occurs, successive READ statements applying to data set reference num-

ber 5 take place on DDNAME FT05F002; therefore, two DDEF commands should be supplied, one for FT05F001 and the other for FT05F002. Similarly, when a WRITE (5) statement is followed by ENDFILE 5, successive WRITE (5) statements take place on DDNAME FT05F002; therefore, two DDEF commands should be supplied, one for FT05F001 and the other for FT05F002.

If a DDNAME with yyy higher than 001 is used, DDEF commands for all yyy between 001 and the higher number must be provided. The various data sets using the same data set reference number may have completely different characteristics and may be stored on the same or different devices.

DSORG=

Specifies the data set organization. In the basic DDEF command, this should be specified as vs (virtual sequential). Other data set organizations and their use are described later in this appendix in "Full DDEF Command."

DSNAME=

The DSNAME parameter description DSNAME=name specifies the name of the data set. This is the name under which the data set is to be cataloged or referred to by other commands during the session. It contains one or more simple names, each simple name having one to eight alphanumeric characters, the first of which must be alphabetic. A period is used as separator between simple names. The maximum number of characters, including periods, is 35. The maximum number of simple names is 18.

For many cases the DSNAME will consist of only one simple name such as:

```
DDEF FT06F001,VS,DSNAME=OUTPUT,DISP=NEW
```

A DSNAME may be of value in describing the contents of the data set. Thus, a program that generates a table of random numbers and a table of square roots with the data set reference numbers 1 and 2, respectively, might employ the DDEF commands:

```
DDEF FT01F001, VS, TABLE.RANNUM
DDEF FT02F001, VS, DSNAME=TABLE.SQRROOTS
```

A simple means of obtaining unique meaningful DSNAMEs is to use the program module name as the first simple name and the DDNAME as the second simple name. Therefore, a program called EXSMOOTH that writes its output on data set reference number 10 might be given DDEF command parameters:

```
DDEF FT10F001,VS,DSNAME=EXSMOOTH.FT10F001.OUTPUT
```

Use of partitioned data set member names and relative generation numbers are for a special kind of data set discussed in detail in this appendix under "Full DDEF Command," but their effect on DSNAMEs is described briefly below.

The DSNAME may continue one of two additional kinds of simple names. The first is written within parentheses and is *not* preceded by a period. The second is a simple name of the form GxxxxVyy, where xxxx is a 4-digit numeric generation number and yy is a 2-digit numeric version.

If a simple name is not separated from the previous name by a period and is within parentheses at the end of a DSNAME, it may be the name of a member of a partitioned data set (first character must be alphabetic) or a relative generation number (zero or a signed integer).

Examples:

MATHLIB(SQRT)	Means the SQRT member of the partitioned data set MATHLIB
PAYROLL(0)	Means the most recent generation of PAYROLL
PAYROLL(-1)	Means the last generation of PAYROLL
PAYROLL(+1)	Means the next generation of PAYROLL
PAYROLL.G0005Y00	Fifth absolute generation

If a DSNAME is to contain generation names, the DSNAME proper is limited to 26 characters, including periods.

Default of DDEF Commands

When an I/O statement is encountered during the execution of FORTRAN programs and the data set reference number is one that has not already been used, the I/O routines make a search of user DDEF commands issued so far. This search is based on the appropriate ddname constructed from the data set reference number according to the convention FTxxFyyy. If no such command is found, the terminal is defaulted.

Conversational

Running conversationally with a defaulted DDEF command means that any WRITE statements in the program cause data to be printed at the terminal. Unformatted WRITE statements result in the printing of hexadecimal data. Formatted or NAMELIST WRITE statements lead to the output of print lines identical to those that would appear on a line printer except that the page skip carriage control character is treated as a triple space and, if the terminal has a print line size shorter than a generated FORTRAN logical record, two or more lines are output (up to a maximum permitted

logical record length of 256 bytes). The first character in the record does not appear but is handled as a carriage control character.

Running conversationally with a defaulted DDEF command means that any READ statement requires input from the terminal.

The expression %END is needed to indicate the end of data to be read by the user's object program from SYSIN. It can be punched on a card for nonconversational processing (or entry through the terminal card reader) or entered through the terminal keyboard. The FORTRAN I/O subroutines detect the end of input by an end-of-data set condition. The system recognizes the expression, %END, and generates an end-of-data set indication, which is transmitted to the FORTRAN subroutines. When a command sequence which has data lines or cards included with it is stored as a SYSIN data set for processing (see Examples 11, 12, and 16), the %END must follow the last line of data.

It is possible to include any number of %END cards in an input deck if the user wishes to section his deck. If the END=option is used in a FORTRAN read statement each %END would then indicate the end of a section of data.

I/O control operations such as REWIND, ENDFILE, and BACKSPACE are ignored when the corresponding DDEF command has been defaulted conversationally.

The I/O statements READ (with no data set reference number) and PRINT lead to automatic default to terminal I/O without prompting for DDEF commands.

Nonconversational

When an I/O statement is encountered during execution of FORTRAN programs in nonconversational mode and the data set reference number has not already been used, a search for the DDEF is made. If no DDEF with the proper data set reference number is found and the I/O operation is WRITE, then output is directed to the system-assigned output data set, SYSOUT. The user receives a listing of SYSOUT, but the data set is not cataloged for subsequent use. If the I/O operation is READ, an attempt is made to read from the SYSIN data supplied by the user when the task was submitted. It should contain data following the RUN command if READ statements without corresponding DDEF commands are to be executed. The characteristics of this data set depend on how it, in turn, was created; it can be either vs or vi and normally contains fixed-length 80-character records. When reading from SYSIN, a record consisting of the characters %END is assumed to be an end-data-set indicator, which causes either termination or transfer to the END=location specified in the READ statement. If the SYSIN data set is exhausted,

(i.e., there is no %END on SYSIN) the same action takes place, but errors will arise if the LOGOFF command has been read as FORTRAN data.

Therefore, data sets and multiple data sets can be read from SYSIN, but it is the user's responsibility to guard against reading commands as if they were data. Further, the user should read all data on SYSIN to avoid that data being interpreted as commands.

I/O control operations such as REWIND, ENDFILE, and BACKSPACE are ignored when the corresponding DDEF command has been defaulted nonconversationally. The I/O statements READ (with no data set reference number) and PRINT lead to automatic default to SYSIN and SYSOUT.

Once a data set has been defaulted to SYSOUT (i.e., a WRITE has been executed where no DDEF was supplied), a subsequent attempt to READ the same data set reference number will cause execution to be terminated.

Full DDEF Command

Those portions of a DDEF command that are applicable to determine or specify the characteristics of a data set operated on by FORTRAN programs are presented in Figure 24. Other parameters and options of the general DDEF command, as described in the publication *Command System User's Guide*, are not given because they are ignored or overridden by the FORTRAN I/O routines.

Specification of DDEF commands for peripheral devices of the CPU is also described in the publication *Command System User's Guide*.

The DDEF command that defines a cataloged data set is brief and simple. The only required operand fields are DDNAME and DSNAME. Other operand fields are unnecessary since other information about the data set is described in its catalog entry. For a cataloged data set if SPACE, UNIT, LABEL, or VOLUME operands are entered, diagnostics will be displayed as appropriate. However, the associated fields will be taken correctly from the existing catalog entry.

DDEF commands that define uncataloged data sets can be divided into two groups: (1) those defining new data sets (data sets that are to be generated during the run but do not yet exist) and (2) those defining old (already existing, but uncataloged) data sets. These old, uncataloged data sets can exist only on private volumes.

To define a new data set that is to be written on a public volume, the user can use the DDNAME, DSNAME, SPACE, DSORG, and LABEL operand fields. Exactly which fields he uses other than DDNAME and DSNAME, which are required, depends on the character of his partic-

OPERATION	OPERAND
DDEF	DDNAME = data definition name [,DSORG = { VI VP VS PS }] ,DSNAME = { data set name * data set name } [,UNIT = ({ DA [,direct-access device type] TA [,tape type] symbolic device address })] [,SPACE = ({ CYL TRK record length } , primary [,secondary] [,HOLD])] [,VOLUME = ([PUBLIC PRIVATE volume sequence number] , [{ PRIVATE volume serial number , ... }])] [,LABEL = [file sequence number] [, { NL SL AL }]] [,RETPD = retention period]] [,DISP = { OLD NEW MOD }] [,OPTION = { CONC JOBLIB }] [,RET = retention code] [,DCB = ([data definition name] [,DSORG=code] [,RECFM=code] [,LRECL=integer] [,BLKSIZE=integer] [,KEYLEN=integer] [,RKP=integer] [,PAD=integer] [,DEVD=code] [,DEN=integer] [,TRTCH=code] [,BUFNO=integer] [,OPTCD= { W A }] [,IMSK=code] [,BFOFF=integer])]]

Figure 24. Full DDEF Command for the FORTRAN User

ular data set. To define a new data set that is to be written on a private volume, the user must give DDNAME, DSNAME, UNIT, and VOLUME operands. If he wants, he can also furnish DSORG, SPACE, LABEL, and DISP fields as well.

The user defines an old, uncataloged data set by specifying the DDNAME, DSNAME, VOLUME, UNIT, and DISP fields. The remaining fields can be defaulted for all data sets except unlabeled tapes.

The description of the basic DDEF command given previously in this appendix also applies to the full DDEF command. If DISP=OLD, the full DDEF command can be used to override data set specifications already given in the standard label; however, the user is cautioned that to do this may cause errors in processing the data.

When DISP=NEW, data sets can be defined that dif-

fer radically from the standard data set resulting from the basic DDEF command. In particular, the user can define output data sets to be made compatible to other systems.

DDNAME

This operand is used exactly the same way as in the basic DDEF command. Refer to "Basic DDEF Command" in this appendix.

DSORG

In the basic DDEF command this is virtual sequential (vs). The other options are virtual index sequential (vi), virtual partitioned (vp),¹ and physical sequential (ps).

¹The DSORG parameter is also present within the DCB sublist of the full DDEF command. This distinguishes between the different forms of VP, namely virtual index sequential partitioned (VIP) and virtual sequential partitioned (VSP), and identifies the organization of the partitioned data set member to be processed.

The `ps` option must be used for tapes or disks that originate outside the `TSS/360` environment and for tapes or disks that are to be written under `TSS/360` and then transferred to other systems for processing.

The data set organization options other than the standard `vs` are available for the benefit of the `FORTRAN` user who wants to process index sequential or partitioned data sets, either to take advantage of their special features or to communicate with assembler language programs.

Each member of a partitioned data set is treated as an independent data set, and the `FORTRAN` user need not be aware of whether it is a member of the data set or not. However, only one `DDEF` command can be issued for a `vp` data set and, therefore, only one member can be processed during a single `FORTRAN` execution.

Virtual index sequential can be used only if there is no `NAMLIST` input/output for the data set and if the user takes the responsibility (for output data sets) of making certain that all logical records contain a sequential key in a specified location. The location and length of this key are given as `RKP=` and `KEYLEN=` within the `DCB` sublist of the full `DDEF` command. `NAMLIST` output cannot be placed on a `vi` data set because there is no way for the user to ensure a sequential key in a given location in every record.

DSNAME

The previous description in this appendix under Basic `DDEF` Command on this subject applies to the full `DDEF` command.

If `DSORG` is `vp`, a member name must be specified as part of the `DSNAME`. No more than one member of a partitioned data set can be processed at one time.

The `*data set name` option of the full `DDEF` command is needed only when processing tape or disk data sets written by the IBM System/360 Operating System with 44-character data set names. Therefore, this option is used only with a `DSORG` of `ps`. Subsequent references to the name do not include the asterisk prefix.

UNIT

This operand is only required when `DSORG` is `ps`. It can be defaulted even in that case if the data set is cataloged.

$$\text{UNIT} = \left(\text{DA}, \left\{ \begin{array}{l} 2311 \\ 2314 \end{array} \right\} \right)$$

Specifies direct-access (either a 2311 Disk Storage Drive or a 2314 Multi-disk Storage Drive).

$$\text{UNIT} = (\text{TA}, \{7|7\text{DC}|9\})$$

Specifies that a tape unit (7-track, 7-track with data conversion, or 9-track) is required for the data set. If given, it should agree with the `DEV` parameter in the `DCB` field.

$$\text{UNIT} = (\text{symbolic device address})$$

Specifies the symbolic device address of a non-standard device.

SPACE

The `SPACE` parameter is never required for existing data sets. It can be used for new virtual storage data sets (`DSORG` is `vi`, `vs`, or `vp`) to request an initial allocation of public storage that is different from that specified at system generation time. Its function in this respect is of interest only if the expected size of the data set is either much larger or much smaller than the standard system allocation. In these cases, it permits somewhat greater efficiency in storage allocation. Even if the storage required is greater than the standard allocation, additional storage is automatically issued so that the `SPACE` parameter is never required for virtual storage data sets.

Form 1

$$\text{SPACE} = (, \text{primary} [, \text{secondary}] [, \text{HOLD}])$$

This form is used to request allocation parameters for virtual storage data sets that differ from the system standard. Primary and secondary allocation are in space units of 4096 bytes (pages). Primary specifies the number of initial space units to be allocated to the data set. It is one to three digits. Secondary is the number of space units to be allocated each time the space allocated to the data set has been exhausted and more data is to be written. This allocation consists of a one- to three-digit decimal number.

The `HOLD` option within the `SPACE` parameter specifies that unused storage assigned to the data set is not to be released when the data set is closed.

Form 2

$$\text{SPACE} = (\{ \text{TRK} | \text{CYL} | \text{record length} \}, \text{primary} [, \text{secondary}] [, \text{HOLD}])$$

This form is used for direct-access devices where `DSORG` is `ps`. It allocates space in units defined by the first subparameter, namely tracks, cylinders, or record lengths.

VOLUME

Form 1

$$\text{VOLUME} = \left(\left\{ \begin{array}{l} \text{PRIVATE} \\ \text{volume serial number} [, \dots] \end{array} \right\} \right)$$

The `VOLUME` parameter is required for old, uncat-

aloged data sets that reside on private volumes. It can also be supplied for new data sets that are to reside on private volumes. Volume serial numbers can be one to six alphanumeric characters and should uniquely identify a particular disk pack or tape reel that is to be mounted. If `PRIVATE` is specified and `DISP=NEW`, the system obtains an available volume and informs the user of the volume selected.

In general, therefore, this form of the `VOLUME` field is needed only for data sets that are not cataloged. It applies mainly when `DSORG` is `PS` and a disk pack or tape generated by the IBM System/360 Operating System is to be read.

Form 2

`VOLUME=(volume sequence number)`

Where a data set extends over more than one volume, specifies the sequence number of the first of the data set to be read or written. This consists of a one-to four-digit number. It is meaningful only if the data set has `PS` organization, is cataloged, and its earlier volumes are not to be processed.

Form 3

`VOLUME=PUBLIC`

This form is used for a new public data set if the user specifies a device type in the `UNIT` parameter. If `PUBLIC` is specified, the volume serial number is not recognized. `PUBLIC` is also assumed if the `VOLUME` parameter is not specified.

LABEL

This parameter applies only when the data set organization is `PS`. It is generally used only when magnetic tapes are to be processed, since all data sets on direct-access volumes have labels known as Data Set Control Blocks (`DSCBS`). The `RETPD` subparameter, however, is applicable to all `PS` data sets.

If the entire `LABEL` field is defaulted, the labeling conventions specified by the installation are assigned. However, if the data set is cataloged, label information is retrieved from the catalog.

Form 1

`LABEL=(file sequence number)`

The file sequence number (one or two decimal digits) specifies the number of a data set on a tape volume containing multiple data sets. By specifying `LABEL`, the user can skip over other data sets and the tape is positioned to the data set he wants. If the user subsequently issues a `REWIND` instruction, the tape will be positioned to the beginning of the data set he is using (not necessarily to the beginning of the tape).

Form 2

`LABEL=(, [NL|SL|AL], RETPD=days)`

The options shown are `NL` for no labels, `SL` for standard labels and `AL` for `ASCII` labels. The exact meaning of standard labels is installation dependent. The `NL` option should not be used for `FORTRAN` output data sets unless a definite reason exists, since a tape data set without labels requires a more complicated `DDEF` command when read back by a `FORTRAN` program than one with labels.

`RETPD` specifies the number of retention days and applies to output tapes with standard labels and to direct-access output.

If defaulted, `RETPD` is set to zero to permit immediate rewriting of any tape or direct access data set.

DISP

`DISP = { OLD
 NEW
 MOD }`

`DISP=OLD` and `DISP=NEW` do not affect a data set's status. Their only function is to guard against use of the wrong data set.

If `DISP=NEW` is explicitly specified, the system verifies that the `DSNAME` does not duplicate one that is already in the user's catalog or one that has been found in a previous `DDEF` command in the same task. If a duplication occurs in conversational mode, the system issues an error message. In nonconversational mode, the task is terminated.

If `DISP=OLD` is explicitly specified, the system searches for an existing data set with the same `DSNAME`. If it cannot find such a data set, it issues an error message; in nonconversational mode, the task is terminated.

If the user does not specify `DISP`, and if the system finds a data set with the specified `DSNAME`, it assumes that it is to use that data set. If it cannot find such a data set, it creates a new data set with that `DSNAME`.

`DISP=MOD` applies only when the data set organization is `PS` and a private volume is being processed. This option causes logical positioning after the last record of the data set. Additional `WRITE` statements are then possible to expand the data set. This option applies mainly to magnetic tapes.

OPTION

`OPTION=CONC`

Specifies that a data set is being added to the concatenated data set named as `DDNAME`. The order of concatenated data sets is the same as the order in which they are defined. *Only existing `PS` data sets can be concatenated.*

`OPTION=JOBLIB`

Specifies that the data set is to be used as a job library. The data set name specified in the `DSNAME` field is entered into the program library list. The data set organization must be `VP`.

RET

The `RET` parameter allows the owner of a virtual storage data set to specify the storage type, and deletion and access attributes of a data set:

`RET=([P|T] [C|L] [U|R])`

The storage types are:

- P – permanent storage
- T – temporary storage
 - if neither is specified, permanent storage (P) is assumed.

The access attributes are:

- C – delete at `CLOSE`
- L – delete at `LOGOFF`
 - if not specified, deletion at `LOGOFF` (L) is assumed for a temporary (T) data set.
 - a permanent data set (P) is not deleted automatically.

The access attributes are:

- U – read/write
- R – read-only
 - if neither is specified, read/write (U) is assumed.

DCB

A data control block (DCB) is one of the major control tables used for communication between `TSS/360` data management and any program requiring control of a data set. For every distinct data set reference number, the FORTRAN input/output routines build a DCB as it is encountered in executing the object program. The DCB is initially void, but can be filled from information in the `DDEF` commands, by the I/O routines after the `DDEF` command has been examined, or by the input data set labels. Therefore, any required information not in the `DDEF` command is entered from one of these sources; in particular, record format (`RECFM`) and logical record length (`LRECL`) take on values determined by the characteristics of an existing data set (`DISP=OLD`) or by FORTRAN input/output standards for `RECFM=VA` where `LRECL` is not required. The only DCB parameters of critical interest to the FORTRAN user are `RECFM` and `LRECL`, and the only time these affect him is when `LRECL` is smaller than a logical input or output FORTRAN record defined within the program. This condition will cause an error message.

Aside from `LRECL` and `RECFM`, the remaining DCB parameters can be grouped into those related to a `DSORG` of `VI` (`KEYLEN`, `RKP`, and `PAD`) and those related to a `DSORG` of `PS` (`DEVD`, `DEN`, `TRTCH`, `OPTCD`, `BLKSIZE`, and `IMSK`).

DCB Parameters – RECFM, LRECL, and BLKSIZE:
`RECFM` specifies the format or character of the records in the data set:

$$\text{RECFM} = \left\{ \begin{array}{l} \text{U[T][A|M]} \\ \text{V[B|T][A|M]} \\ \text{F[B|T|BT][A|M]} \\ \text{D[B]} \end{array} \right\}$$

Where the record format is:

- U – undefined-length records
 - if `DSORG` is `VS` or `VI`, this means record size is 4096 bytes
 - if `DSORG` is `PS`, this means records have physical boundaries (on a tape) and can vary in length, also known as blocksize.
- V – variable-length records (EBCDIC)
 - each record contains in the first four bytes a binary count of the number of bytes in the record
 - maximum data byte limit is:
 - 1,048,572 bytes for `VS`
 - 4,000 bytes for `VI`
 - 32,756 bytes for `PS`
 - maximum data byte limit for formatted records with `VS` is 4092 bytes.
- F – fixed-length records
 - maximum record length is:
 - 1,048,576 bytes for `VS`
 - 4,000 bytes for `VI`
 - 32,760 bytes for `PS`
 - maximum formatted record length with `VS` is 4092 bytes.
- D – variable-length records
 - for ASCII tapes only
 - can be specified only as a DCB subparameter of `DDEF` command.

Where the physical attributes are:

- B – blocked records (meaningful only for `PS`); the maximum blocksize is 32,760 bytes
 - `BLKSIZE` must be an integral multiple of logical record length for format-F records
- T – track overflow employed
 - applies only to disk data sets with `DSORGPS`
 - may cause errors if omitted when track overflow is to be used for writing very long records.

`LRECL` must be 4 bytes greater than the largest formatted logical FORTRAN record that is to be read or written. If defaulted for format-V records, it is assumed to be 4096 for `VS` and 4000 for `VI`. If larger `VS` records are anticipated, it must be specified by the user. Format-V records with a `DSORG` of `PS` are processed with an assumed `LRECL` of 4096, which can also be raised by the user.

When using `VSAM`, the FORTRAN `BACKSPACE` statement may not be used to backspace a data set of undefined format (`RECFM=U`).

The A or M options under `RECFM` relate to whether extended ANSI FORTRAN control characters or machine code control characters appear as the first byte of every record. The A option (default value) is usually preferred, since most formatted FORTRAN records make use of extended ANSI FORTRAN control character conventions. If `NAMELIST` records are written, the A can be specified (or chosen by default) since all records start

with a blank. The M option should not be specified unless the user has coded hexadecimal data into the first byte of every record. This option can be ignored by most users.

BLKSIZE is required only if RECFM is FB (fixed-length blocked records) and this option, in turn, is meaningful only if DSORG is PS. In this case, it must be a multiple of LRECL. Otherwise, any value given is ignored and replaced by LRECL.

Examples of how to use the RECFM, LRECL, and BLKSIZE parameters are shown below (they are not complete DDEF commands; only the DSORG and DCB portion is shown).

VS, DCB=(RECFM=F, LRECL=80)	VS 80-character fixed-length records
VS, DCB=(RECFM=V)	VS variable-length records (standard)
VS, DCB=(RECFM=FA, LRECL=133)	VS data set for listing
VI, DCB=(RECFM=V, RKP=4, KEYLEN=4)	VI variable-length records with 4-byte key in initial position (after 4-byte control word)
PS, DCB=(RECFM=FB, LRECL=100, BLKSIZE=1000)	PS fixed-length blocked with 10 records per block

DCB Parameters – VI Associated: If DISP=NEW and DSORG is VI, the user must specify record key position (RKP), key length (KEYLEN), and optionally padding percent (PAD). The relative key position (RKP) specifies the displacement of the key field from the first byte of the logical record. Since VISAM records cannot exceed 4000 bytes, any value between 0 and 3999 can be specified. The first four bytes of a format-V logical record are reserved for length information. Therefore, if RECFM=V, RKP must be four bytes higher ($4 \leq \text{RKP} \leq 3999$) than for the same data set with RECFM=F. KEYLEN is the length in bytes of the key associated with a record. The maximum value is 255.

PAD specifies the percent of space (to a limit of 50 percent) to be left available within the pages of a VI data set, thus providing for insertions within pages. The FORTRAN user can use VI data sets as output provided he ensures that the specified key field is maintained with appropriate data in collating sequence (ascending) from record to record. This is most easily done by specifying an integer output field as in the following example of a FORTRAN program for reading input records and copying them to a VI data set.

```

1  FORMAT (1X, I5, 6A4)
2  FORMAT (6A4)
   DIMENSION A(6)
   I=0

```

```

3  READ (1, 2, END=4) A
   I=I+1
   WRITE (2, 1) I, A
   GO TO 3
4  STOP
   END

```

DDEF commands are:

```

DDEF FT01F001, , DSNAME=INPUT, DISP=OLD
DDEF FT02F001, VI, DSNAME=OUTPUT,
   DCB=(RKP=1, KEYLEN=5, RECFM=F,
   LRECL=30), DISP=NEW

```

The user cannot write NAMELIST records on a VI data set. The user should exercise caution in maintaining the key when writing unformatted records on VI data sets.

DCB Parameters – PS Associated: If DSORG is PS, a large number of DCB parameters can be used that otherwise have no meaning. As previously discussed, the BLKSIZE parameter is required if RECFM=FB. In addition, the parameters listed below apply.

1. *DEVD:* Specifies the device on which the data set resides. It is not required for cataloged data sets; it can be one of the following:
 - a. DA specifies direct-access (disk formatted in accordance with IBM System/360 Operating System conventions). In this case KEYLEN has a special meaning since it specifies how many of the initial bytes of each record are to be written on the disk (or read from it) as a key. This condition has no connection with VI and the key cannot be used for random access by the FORTRAN user. If all processing is to be done on TRS/360, it is not necessary to use it. However, if a data set is to be written on a disk pack for the purpose of being processed on another IBM system (e.g., IBM System/360 Operating System), the use of KEYLEN may be required.
 - b. TA specifies magnetic tape. If 7-track tape is specified in the UNIT parameter, DEN is given a value of 0, 1, or 2 for recording density of 200, 556, or 800 bytes, respectively. If 7-track tape is to be read, TRTCH can be given as C for data conversion, E for even parity, and T for BCDIC to EBCDIC conversion. The defaults are odd parity and no translation.
2. *BUFNO=I:* Physical sequential I/O normally takes place with two buffers. The user can reduce space allocation requirements by specifying the number of buffers as only one. Any other value given to BUFNO is disregarded.
3. *OPTCD=W:* Applies only for direct-access output, causes additional checking of all write operations. This will increase execution time. OPTCD = A is specified for an ASCII tape.

4. *IMSK*=code: Specifies a 4-byte hexadecimal number whose bit pattern indicates the system's error-handling procedures to be invoked. If *FFFFFFFF* is written, the system is to apply all optional error recovery procedures. This is the default condition. If *00000000* is written, the system is to apply none of its optional error recovery procedures. If any other 4-byte hexadecimal number is written, the system applies its error-recovery procedures whenever a bit is set to one in *IMSK* which corresponds to an error condition. The first two bytes correspond to the first two bytes of the channel status word, and the other two correspond to the first two sense bytes. Bit positions in each byte for specification of system error recovery procedure are in the following format:

```
XXXXXXXXB XCXXXXXD YEFGHIYY YYYYYYYY
```

where a 1-bit in a given position indicates that the system is to handle the associated error condition.

X = System never tests this bit to determine entry to retry routines

Y = Device-dependent conditions

B = Unit exception

C = Incorrect length

D = Channel chaining check

E = Intervention required

F = Bus-out parity

G = Equipment check

H = Data check

I = Overrun

DDEF Summary

Table II shows, for each type of data set the required and optional parameters for the *DDEF* command. The major category is device type (direct-access or tape), followed by disposition (new or old) for direct-access device. Data sets residing on magnetic tape are grouped as cataloged (*CAT*) or uncataloged (*UNC*), and labeled (*LAB*) or unlabeled (*UNL*).

Sample DDEF Commands

Commands are presented here in increasing order of complexity and decreasing order of likely applicability.

1. *DDEF FT01F001,VS,SCRATCH*

Can apply to a program containing such statements as *WRITE (1)*, *READ (1)*, *REWIND 1*, etc., on a temporary (scratch) data set.

The first parameter, *FT01F001*, is built according to the skeleton *FTxxFyyy*, where *xx* is the data set reference number used in the program (01 in this case) and *yyy* is a sequence number (generally 001).

vs indicates virtual sequential.

The *DSNAME* can be any valid name made up of eight-letter components connected with periods.

2. *DDEF FT03F001,DSNAME=SCRATCH.ONE*
DDEF FT04F001,DSNAME=SCRATCH.TWO

Illustrates the use of two scratch data sets, distinguished by having different *DSNAMES*.

3. *DDEF FT05F001,VS,DSNAME=SCRATCH,DISP=NEW*

This may be used if the data set named *SCRATCH* is either not yet in existence or has not been cataloged.

4. *DDEF FT06F001,VS,DSNAME=SCRATCH,DISP=OLD*

This may be used if the data set named *SCRATCH* is already in existence and has been cataloged.

5. *DDEF JLIB,VP,DSNAME=OWNER.JOBLIB,OPTION=JOB-LIB*

This would never be used with object time I/O but can be applied when the user wants to compile programs onto a job library other than the standard *USER-LIB* that *LOGON* supplies.

DISP is defaulted to *NEW* only on the first use of the data set *OWNER.JOBLIB*. Subsequent action (compiling new members, etc) requires the same *DDEF* command but *DISP* is defaulted to *OLD*. (Job libraries are described in Appendix C.)

6. *DDEF FT07F001,VS,DSNAME=OUTPUT.F120,DCB=(RECFM=F,LRECL=120)*

Creates a new *vs* data set with fixed-length records of 120 bytes. The *DSNAME* can be anything; a suggested technique is to incorporate information defining the kind of data set.

The *DCB* parameters appear within parentheses and consist of *RECFM*, meaning record format, and *LRECL*, meaning logical record length.

If *DISP=NEW* and no *DCB* parameters are given, it is assumed that *DCB=(RECFM=V,LRECL=4096)*. If *DISP=OLD*, the data set characteristics that appear in the data set label are used.

The above *DDEF* command should *not* be used with *DISP=OLD*, since the *DCB* parameters may not be correct.

7. *DDEF FT08F001,VS,DSNAME=OUTPUT.F120,DISP=OLD*

This is the correct way to read back the data set created by the last example.

8. *DDEF FT09F001,VS,DSNAME=DATA.V,DCB=(RECFM=V,LRECL=4096)*

This has the same effect as *DDEF FT09F001,DSNAME=DATA.V*, since *V* and 4096 are the default values supplied for *RECFM* and *LRECL* by the FORTRAN I/O routines.

9. *DDEF FT10F001,VI,DSNAME=LIST.VIF80,DCB=(RECFM=F,LRECL=80,RKP=0,KEYLEN=5)*

VI indicates virtual index sequential. This can be omitted if the installation-defined option is *VI*.

Table 11. DDEF Parameter Requirements by Data Set Type

	Direct Access										Tape										
	New Data Set					Old Data Set					Old			New or Old UNC UNL							
	VS	VI	VP	PS	CAT any dsorg	UNC		CAT LAB	CAT UNL	UNC LAB											
						VS VI VP	PS														
DDNAME	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
DSORG	VS	VI	VP	PS	10	X	PS							PS				PS			
DSNAME	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
DCB	DSORG																				
		F	V	U	F	V							F	V	U			F	V	U	
	RECFM	X	*	X	X	*							X	X	X			X	X	X	
	LRECL	X	7		X	8							X	2				X	2		
	BLKSIZE												1					1			
	KEYLEN				X	X							4	4	4			4			
	RKP				X	X															
	PAD				0	0		S	S												
	BUFNO							A	A	0	0	0			0	0	0	0	0	0	0
	OPTCD							M	M	0	0	0			9						
	DEVD							E	E	DA	DA	DA			DA			TA	TA	TA	TA
	DEN																	0	0	0	0
	TRTCH							A	A									0	0	0	0
IMSK							S	S	0	0	0			0	0	0	0	0	0	0	
UNIT									X	X	X			X			X	X	X	X	
SPACE	0	0	0	0	0		V	V	9	9	9										
VOLUME	3	3	3	3	3		I	S	X	X	X			X	X		X	X	X	X	
LABEL									6	6	6						5	5	5	5	
DISP									*	*	*			X	X	X	X	X	X	X	
OPTION														0	0	0	0	0	0	0	
RET	0	0	0	0	0																

- Notes:
1. Required for RECFM=FB. BLKSIZE must be a multiple of LRECL.
 2. Specify largest V type record expected. Default values are as described in this appendix.
 3. Specify only if private volume desired.
 4. Refer to "DCB Parameters - PS Associated" in this appendix.
 5. Specify NL for unlabeled data sets. A file sequence number may be specified.
 6. Only retention period (RETPD) may be specified.
 7. Default is 4092 bytes.
 8. Default is 4000 bytes.
 9. Only meaningful if existing data set being extended.
 10. Required for nonVAM data sets.

Legend: Shaded - Field not required in DDEF command
 X - Field required in DDEF command
 0 - Field optional
 * -IBM-supplied default

The additional DCB parameters define a new data set having a sequential key in every record in the first five bytes. The user must make sure that the key is in ascending sequence (e.g., by using a format such as FORMAT (15, . . .) and incrementing the key in the output list before each write).

RKP means record key position relative to the first byte of the record. KEYLEN means key length in bytes.

10. DDEF FT11F001,VI,DSNAME=LIST,VIV,DCB=(RECFM=V,RKP=4,KEYLEN=5)

This differs from sample 9 only in that it is for variable-length records. LRECL can be omitted or set as

high as 4000, which is the maximum possible value for VI.

RKP=4 because each logical record has a four-byte length field that is not maintained by the user but supplied automatically by the FORTRAN I/O routines; it is followed by a five-byte sequential key that is maintained by the user.

11. DDEF FT12F001,VI,DSNAME=LIST,VIF80
 DDEF FT13F001,VI,DSNAME=LIST,VIV

The above DDEFs might be used to read back the data sets created by 9 and 10, respectively.

Note how characteristics are defaulted. They will

be filled in from the catalog and input data set labels. Note also that DISP in this case is defaulted to OLD.

12. DDEF FT19F001,PS,DSNAME=TAPE.OS360,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DEV=TA),UNIT=(TA,9),VOLUME=(PRIVATE),LABEL=(,SL,RETPD=50)

A DDEF such as this might be used to instruct the operator to mount a scratch tape on a 9-track drive (the volume identification is sent to the user after entry of the DDEF command). It is written with blocked 80-character records, at 10 records per block. The density, DEN, and TRTCH DCB subparameters do not apply since 9-track tape is specified.

Assuming the volume serial number assigned is 00896, the next DDEF may be used to read the tape back into a FORTRAN program.

13. DDEF FT20F001,PS,DSNAME=TAPE.OS360,VOLUME=(,000896),UNIT=(TA,9)

DCB parameters are not needed if the installation default for LABEL is (,SL) which has been assumed in this example.

14. DDEF FT20F001,PS,DSNAME=TAPE.OS360

This assumes that when 12 was created it was cataloged. The act of cataloging a data set makes it much easier to write DDEF statements for retrieving it.

15. DDEF FT21F001,VS,DSNAME=PRIVATE.VAM,VOLUME=(PRIVATE)

VAM data sets may be written on private volumes, provided the volumes are correctly formatted. The actual identification of the volume used is sent back to the user. Data set characteristics by default will be RECFM=V,LRECL=4096.

16. DDEF FT22F001,VS,DSNAME=PRIVATE.VAM

This reads back the data set created by 15. It is assumed that it has been cataloged.

17. DDEF FT23F001,PS,DSNAME=DISK.OS360,DCB=(RECFM=FB,LRECL=40,DEV=DA,BLKSIZE=400),UNIT=(DA,2311),VOLUME=(PRIVATE),SPACE=(400,500,20)

A data set is to be written on a PS-formatted direct-access volume to be mounted for private use. Space is to be reserved for 500 blocks of 400 bytes each, each block containing 10 logical records of 40 bytes. Each secondary allocation is to be 20 blocks of 400 bytes each. The device type is 2311.

18. DDEF FT24F001,PS,DSNAME=DISK.OS360

This reads back the data set created in 17. It is assumed that it has been cataloged.

19. DDEF FT25F001,PS,DSNAME=DISK.OS360
VOLUME=(,05789),UNIT=(DA,2311)

This is the same as 18, but is used if the data set had not been cataloged. The operator must be informed of the volume identification and device type. All other data set characteristics will be obtained by the system from standard direct-access labels, which must be present on all direct-access volumes.

20. DDEF FT26F001,PS,DSNAME=BADSYS.TAPE,DCB=(RECFM=F,LRECL=80,DEV=TA),UNIT=(TA,9),
LABEL=(,NL),VOLUME=(PRIVATE)

This is not recommended unless the tape is to be used by another system that cannot process standard tape labels.

A tape reel is to be mounted and the user informed of its identification. Since no labels are written, the characteristics of the data set and its name are not apparent to any program reading the tape back.

21. DDEF FT27F001,PS,DSNAME=BADSYS.TAPE,DCB=(RECFM=F,LRECL=80,DEV=TA),UNIT=(TA,9),
LABEL=(,NL),VOLUME=(009876)

This is needed to read back the tape made by 20 (assuming that volume 009876 was assigned to it). Only DISP=OLD can be defaulted since there are no tape labels and it was not cataloged.

22. DDEF FT28F001,PS,DSNAME=BADSYS.TAPE,DCB=(RECFM=F,LRECL=80),DISP=OLD

This is needed to read back the tape made by 20, provided it has been cataloged. DCB parameters are still needed, but VOLUME, UNIT, DEV, and LABEL parameters are supplied from the system catalog.

23. DDEF FT29F001,VS,DSNAME=MULTIPLE.PART1
DDEF FT29F002,VS,DSNAME=MULTIPLE.PART2
DDEF FT29F003,VS,DSNAME=MULTIPLE.PART3

The three DDEF statements apply to a program that executes the following sequence of statements:

```
WRITE(29) . . . ENDFILE 29 . . . WRITE(29) . . . ENDFILE 29 . . . WRITE(29)
```

24. DDEF FT30F001,VS,DSNAME=MULTIPLE.PART1.V
DDEF FT30F002,VS,DSNAME=MULTIPLE.PART2.F,DCB=(RECFM=F,LRECL=100)
DDEF FT30F003,VI,DSNAME=MULTIPLE.PART3.VIF80,
DCB=(RECFM=F,LRECL=80,RKP=0,KEYLEN=5)

This is similar to 23, except that the three data sets have different characteristics.

25. DDEF FT31F001,,DSNAME=MULTIPLE.PART1
DDEF FT31F002,,DSNAME=MULTIPLE.PART2
DDEF FT31F003,,DSNAME=MULTIPLE.PART3

This is similar to 23, but applying to a program that reads the three data sets back as follows:

```
1 READ (31, END=2)
  .
  .           [Process MULTIPLE.PART1]
  GO TO 1
2 READ (31, END=3)
  .
  .           [Process MULTIPLE.PART2]
  GO TO 2
3 READ (31, END=4)
  .
  .           [Process MULTIPLE.PART3]
  GO TO 3
4 STOP
```

Error Messages for the DDEF Command

The user's replies to diagnostic messages issued for his DDEF command should be guided by the following:

1. If the diagnostic message calls for reentering an element within a given operand field, only that element should be reentered. Preceding and/or following delimiters are unnecessary. Default is acceptable.
2. If the diagnostic message calls for reentering a complex operand field, the whole field should be reentered, including keyword and equal sign. Default is acceptable.
3. If the diagnostic message calls for reentering an operand field that consists of only one element in addition to the keyword, the reply may be either the element alone, or the keyword, equal sign, and element.
4. If the diagnostic message calls attention to an inconsistency and asks the user to (re)enter one of two or three specified operands, the reply must be a complete operand field. Default is acceptable only if so stated in the message.

The user is informed if the DDEF command cannot be completed. This can occur for one of the following reasons:

1. Invalid punctuation in the operand string.
2. User's volume(s) cannot be mounted.
3. Sufficient space cannot be allocated.
4. More than three logical inconsistencies were detected in the DDEF command.

Whenever possible, correction and completion of the command is attempted. But if diagnostic messages indicate that an operand has been misunderstood because of a punctuation error in the operand string, the user should interrupt the operation (by pressing the attention key) and reenter the corrected command.

The user must never reenter an operand or part of an operand that has not been requested.

If a keyword is missing or invalid, the pertinent element following it must be reentered after the corrected keyword and equal sign are typed.

If an operand occurs twice in the operand string, the second occurrence is preferred. All elements belonging to the earlier occurrence are erased.

DDEF prompting messages are issued according to the operand information already supplied. Unnecessary prompting is kept to a minimum.

If the user's problem program is being executed in conversational mode and an undefined DDNAME is referenced, prompting messages for DDEF operands are issued to the user.

Data Set Definition Rules for Language Processing

Table 12 provides information relating to the organization of and DDEF requirements for data sets involved in assembly, compilation, and linkage editing.

Data Set Definition Rules for TSS/360 Commands

Table 13 provides information relating to the structure of and DDEF requirements for data sets processed by TSS/360 commands.

Table 12. Data Set Definition Rules for Language Processing

COMMAND	RELATED DATA SETS	DATA SET DEFINITION RULES	DSORG
FTN (FORTRAN)	Source program data set	Source program data sets: If supplied as part of SYSIN data set, these data sets do not require any further data set definition. If supplied as prestored data set, they must be cataloged. No DDEF is required for these commands.	VI
	Object module		
	Listing data set		
ASM (ASSEMBLER)	Source program data set	Object module: The module is placed in library at top of program library list. If job library is to receive object module, DDEF command is required to define library.	VP
	Object module		
	Listing data set	Listing: No DDEF command required.	VI
LNK (LINK EDITOR)	Source program data set	Same rules as for FTN and ASM.	
	Libraries that are to supply object modules	Each library referred to by INCLUDE statements except USERLIB and each job library used by automatic call must be defined by a DDEF command.	VP
	Library to receive output object module	If library at top of program library list is to receive output object module, no additional DDEF in this task. If another library is to receive output, it must be defined by previous DDEF command and be specified by its DDNAME to linkage editor.	VP
	Listing data set	No DDEF command required.	VI

Table 13. Data Set Definition Requirements for Commands

COMMAND	RELATED DATA SETS	DSORG	DATA SET DEFINITION
BACK	New SYSIN data set that is to control completion of this task in non-conversational mode.	VS, VI	New SYSIN data set must be cataloged, or defined by previous DDEF command in conversational portion of this task.
CATALOG	Data set to be cataloged.	PS	Data set to be cataloged must be defined by previous DDEF command in this task, unless UPDATE option specified.
CDD	Data set containing only DDEF commands.	VI	Data set must be cataloged, or defined in current task.
CDS	Data set to be copied; existing data set or member of partitioned data set.	VS, VI	Data set to be copied must be cataloged or defined by previous DDEF command in this task.
	Copy data set: can be data set or member of partitioned data set.	VS, VI	Copy data set is defined by this command.
DATA (See Note 2)	Data set to be entered.	VS, VI	No DDEF command is required if the data set is to reside on public storage; data follows this command in input stream. If the data set is to reside on private storage, a DDEF must be issued before the command.
DEFAULT	User profile data set in USERLIB.	VP	Data set must be defined in current task.
DELETE	Data set whose name is to be removed from catalog.	any	No DDEF command required for this command.
DSSP	Data sets whose status is desired.	any	Each data set whose status is to be presented must be cataloged; no DDEF command required for this command.

Table 13. Data Set Definition Requirements for Commands (continued)

COMMAND	RELATED DATA SETS	DSORG	DATA SET DEFINITION														
DUMP	Data set to be printed as a result of program control command DUMP.	VI	DDEF command whose DDNAME is PCSOUT must be defined prior to execution of DUMP command.														
EDIT (See Note 1)	Data set to be processed by the Text Editor.	VI	Data set must be cataloged, defined in current task, or defined by this command.														
END (See Note 2)	Data set being processed by the Text Editor, or indicates PROCDEF Command completion.	VI	No DDEF command required for this command.														
ERASE	Data set to be erased.	VS, VI, VP	Data set to be erased must be cataloged.														
EVV	Data sets whose names are to be entered in the catalog.	VS, VI, VP	No DDEF command required by this command.														
EXECUTE	SYSIN data set for nonconversational task set up by this command.	VS, VI	Data set must be cataloged; no DDEF command required by this command.														
LINE?	Line data set containing lines to be presented.	VI	Line data set must be cataloged or defined by previous DDEF command in this task.														
LOAD	Object module to be loaded.	VP	Object module to be loaded is identified by external name specified in this command; it must be in a library in the current program library list.														
	<table border="0"> <tr> <td style="font-size: 3em; vertical-align: middle;">}</td> <td style="padding-left: 0.5em;">FTN</td> </tr> <tr> <td></td> <td>ASM</td> </tr> <tr> <td></td> <td>LNK</td> </tr> <tr> <td></td> <td>User-</td> </tr> <tr> <td></td> <td>written</td> </tr> <tr> <td></td> <td>problem</td> </tr> <tr> <td></td> <td>program</td> </tr> </table>	}	FTN		ASM		LNK		User-		written		problem		program		
}	FTN																
	ASM																
	LNK																
	User-																
	written																
	problem																
	program																
MODIFY	Data set to be changed.	VI	Data set must be cataloged or defined by previous DDEF command in this task.														
PC?	Data sets whose status is desired.	any	Each data set whose status is to be presented must be cataloged; no DDEF command required for this command.														
PERMIT	Data sets for which sharing is permitted.	any	Data sets for which sharing is permitted must be cataloged; no DDEF command required for this command.														
POD?	Virtual partitioned data set for which information about its members is given.	VP	Virtual partitioned data set must be cataloged, or defined by previous DDEF command in this task.														
PRINT	Data set to be printed.	PS, VS, VI	Data set must be cataloged or defined by previous DDEF command in this task. Data sets on unlabeled tapes must be defined by a DDEF command.														
PROCDEF	Data set which consists of other commands, to become a user-written procedure.	VI	Data set must be defined in current task.														
PROFILE	User profile data set in USERLIB, session profile in task virtual storage.	VP	Data sets must be defined in current task.														
PUNCH	Data set to be punched on cards.	VS, VI	Data set must be cataloged or be defined by previous DDEF command in this task.														
REGION (See Note 1)	Data set to be processed by the Text Editor.	VI	Data set must be cataloged, or defined in current task.														
RELEASE	Data set to be released.	any	Data set to be released must be defined in previous DDEF command in this task.														
RET	VAM data set whose data set descriptor is to be changed.	VS, VI, VP	Data set must be cataloged.														

Table 13. Data Set Definition Requirements for Commands (continued)

COMMAND	RELATED DATA SETS	DSORG	DATA SET DEFINITION
SHARE	Data sets for which sharing is requested.	any	Data sets for which sharing is requested must be cataloged; no DDEF command required by this command.
SYNONYM	User profile data set in USERLIB, session profile in task virtual storage.	VP	Data sets must be defined in current task.
TV	Physical sequential data set (from a VT operation) to be written on a VAM volume.	PS	Data set (input) must be cataloged, or defined in current task.
VT	VAM data set to be copied to magnetic tape as a physical sequential data set.	VS, VI, VP	Data set (input) must be cataloged or defined in current task.
VV	VAM data set to be copied into direct access storage.	VS, VI, VP	Data set (input) must be cataloged, or defined in current task.
WT	Data set to be recorded on magnetic tape in print format.	VS, VI	Data set must be cataloged or defined by previous DDEF command in this task.

Note 1: These are the basic directive commands of the Text Editor. See *Command System User's Guide* for details concerning the data manipulation commands of this facility.

Note 2: If the DATA command was used to create the data set within the current task, then the data set is defined as if a DDEF command had been issued by the user directly. If the data set is also VAM organized and resides in public storage, it is automatically cataloged.

Appendix F. Attention Considerations

Interrupting Execution

Pressing the `ATTENTION` button on the terminal lets the user interrupt the execution of programs within his task. The effect of the `ATTENTION` interruption depends upon the privilege class of the interrupted module, the nature of the module (some privileged modules are sensitive to attention interruptions and some are not), and even the language in which the source program is written.

Interrupting Privileged Commands

If `ATTENTION` is pressed during the execution of a privileged command imbedded within a command string (one or more commands in one `sysin` line), the system responds by printing an asterisk (*) at the terminal, meaning "more commands remain to be processed." To display a list of remaining commands, the user responds by issuing `STRING` (see Table 14). All commands processed during or before the attention interruption can be assumed to have successfully executed if the system issues no diagnostic message. To resume execution of the remaining commands, the user may press `RETURN`, or he may ignore the remaining commands by issuing new commands unrelated to the interruption.

If `ATTENTION` is pressed during the execution of a singly issued privileged command, or during execution of the last command in a command string, the command will normally complete execution and prompt the user to enter another command. If the `ATTENTION` interruption prevents the command from completely executing, however, the system issues a diagnostic message.

Interrupting Nonprivileged Commands and User Programs

If `ATTENTION` is pressed during the execution of a nonprivileged command (all language processors, for example, are nonprivileged), or during the execution of a user's program (including `FORTRAN` library subprograms), the system responds by printing an exclamation point (!) at the terminal. The user may invoke `PCS` commands to display critical fields and modify values (see "Using the Program Control System," in this appendix), or, to resume processing at the point

of interruption, he may issue `GO` (or press the `RETURN` key).

Attention Levels

When a user stops program execution by pressing `ATTENTION`, the status of the interrupted program is saved and can be restarted later at the point of interruption by issuing the `GO` command. Privileged commands cannot be restarted, however.

Nonprivileged commands (`FTN` or `EDIT`, for example) and user-written programs coded in assembler language or in `PL/1` can be interrupted and saved at 10 levels: each time a nonprivileged program or command is interrupted by `ATTENTION`, its status is saved so that it can be restarted later.

As many as 10 such programs can be interrupted and saved for later execution (10 `ATTENTION` levels). `ATTENTION` interruptions of more than 10 nonprivileged programs (that is, a request for more than 10 `ATTENTION` levels) will cause the status of the earliest-saved level to be lost.

`FORTRAN` programs interrupted by `ATTENTION` can be saved, too, but only one `FORTRAN` program can be saved: calling a second `FORTRAN` program will cause a `FORTRAN` program previously saved at any level to be lost.

Using The Program Control System (PCS) with ATTENTION

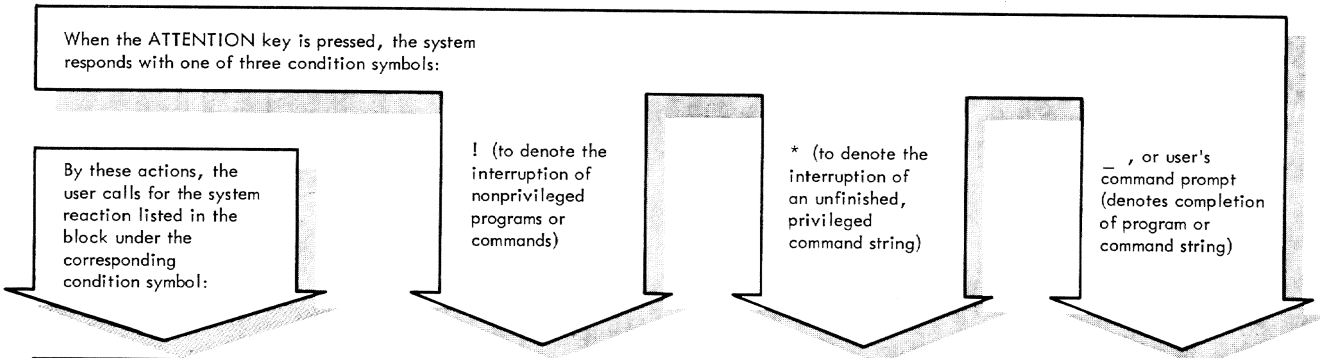
User-written programs may be interrupted during execution by pressing the `ATTENTION` key. The interrupted program and all related data sets are frozen when the attention interruption is received. Using `PCS` commands, the user can inspect any portion of the object module, he can display or modify the values of variables, registers, or object code, and resume execution using the values he has modified.

See Appendix B, "PCS and `FORTRAN` Object Programs," in this book, for further information about using the Program Control System. For complete information about using `PCS` commands, see *Command System User's Guide*.

Responding to Attention Interruptions

Table 14 shows how the user can make full use of the multi-level `ATTENTION` handling capability of `TSS/360`.

Table 14. Responding to Attention Interruptions



By issuing the GO command . . .	the current user program is resumed	the most recently interrupted user program is resumed and intervening command strings are cancelled	
By issuing any command . . .	the command is executed	the command is executed and the current command string is cancelled	the command is executed
By pressing the RETURN key . . .	the current user program is resumed	the current command string is resumed	the system prompts the user to enter a command
By pressing the ATTENTION key . . .	the system returns an exclamation point (nothing is changed)	the system returns an asterisk (nothing is changed)	the system prompts the user to enter a command
By entering STRING to list remaining commands in an interrupted string . . .	the system displays the unexecuted command string, if it exists	the system displays unexecuted commands in the current command string	the system returns a diagnostic message
By entering STACK to list names of active nonprivileged programs (in the order in which they may be retrieved). . .	the system displays the names of active user programs at every ATTENTION level		
By entering EXIT to end the currently active program . . .	ends the currently active program, resumes command string if it exists	ends the most recently interrupted program and resumes its associated command string, cancelling subsequent command strings	
By entering RTRN to cancel command strings and user programs at every attention level . . .	command strings and user programs are cancelled by the system at every ATTENTION level		
By entering PUSH to save the status of the currently active program	the system saves the status of the currently active program in ISA Long Save 1 (ISALS1)		

Appendix G. Command Formats

This appendix explains the notation used in command descriptions and presents the general form of each rts/360 user's command.

The command language statements given in this appendix and in Appendixes A and E use the conventions given below in their notation.

Operands

A positional operand is represented by:
value-mnemonic or operand-name

In the first case, the user writes *only* a value of one of the forms specified by the value mnemonic. In the second case, the operand name is merely a means of referring to the operand in the format description; the hyphen simply separates elements of the operand description and is not written in the actual operand.

A keyword operand is represented by:

KEYWORD=value-mnemonic or KEYWORD=operand
name

The user first writes the keyword and the equal sign, and then either a value of one of the forms specified by the value mnemonic, or an operand name, as required.

The following general rule applies to the interpretation of operand representations in a format description: when the operand is written, anything shown in upper-case letters must be written exactly as shown; anything shown in lower-case letters is to be replaced with a value provided by the user. Thus, in the case of a keyword operand, the keyword and equal sign are written as shown, and the value mnemonic is replaced. In the case of a positional operand, the entire operand representation is replaced.

Some operands are not represented in format descriptions by operand names or value mnemonics. Instead, they are represented by one or more upper-case character strings that show exactly how the operand should be written. These character strings are called coded values, and the operands for which they are written are called coded value operands.

A coded value operand results in either a specific value parameter or a specific sequence of executable instructions.

When a positional operand can be written as only one coded value, the operand is shown simply as the

coded value; an additional lower-case operand name is not used. For example, a positional operand could be represented by:

MYDATA

A keyboard operand could be represented by:

KEYWORD=MYDATA

If a positional operand can be written as any one of two or more coded values, an additional lower-case operand name may or may not be used. The choice of which is done is determined by whether or not a name can be meaningfully used to refer to all values of the operand. For example, a positional operand could be shown as either of the following:

{NL|SL}
mode-{NL|SL}

In both of the above examples, the braces indicate that the coded values are grouped together in one operand representation, and the vertical stroke indicates that either one of the coded values can be written. The braces and vertical strokes are metasymbols.

Metasymbols

Metasymbols are symbols that convey information to the programmer, but are not written by him. They assist in showing the programmer how and when an operand should be written. The metasymbols used in this publication are:

1. | This is a vertical stroke and means "or." For example, A|B means either the character A or the character B. Alternatives are also indicated by being aligned vertically (as shown in the next paragraph).
2. {} These are braces and denote grouping. They are used most often to indicate alternative operands. For example:

{NL|SL}
{NL}
{SL}

The two examples above are equivalent; either NL or SL must be written.

3. [] These are brackets and denote options. Anything enclosed in brackets can be either omitted

or written once in the command. For example:

```
[NL]
[NL|SL]
[ NL ]
[ SL ]
```

The second and third examples above are equivalent; NL or SL or neither can be written. The underlining indicates that, if neither is written, SL is assumed. Braces used for grouping inside brackets are redundant.

4. . . . This is an ellipsis. It denotes occurrence of the preceding syntactical unit one or more times in succession. A syntactical unit is any combination of operand representations, commas, parentheses, and metasympols, enclosed in braces. For example:

{symbol,} . . .

The above example indicates that a symbol followed by a comma can be written any number of times, but it must be written at least once. The braces denote grouping, and are the extremities of the syntactical unit to which the ellipsis refers.

General Forms

Each of the TSS/360 user's commands is shown below in its general form. Detailed specifications for these commands are given in *Command System User's Guide*; this list is intended for quick reference only. Note that only the basic form of the DDEF command is given; the full form appears in Appendix E.

OPERATION	OPERANDS
ABEND	

OPERATION	OPERANDS
ABENDREG	

OPERATION	OPERANDS
ASM	NAME=module name [,STORED= {Y N}] [,MACROLIB=({data definition name of symbolic portion, data definition name of index portion} [, . . .])] [,VERID=version identification] [,ISD={Y N}] [,SYMLIST={Y N}] [,ASMLIST={Y N}] [,CRLIST={Y N}] [,STEDIT={Y N}] [,ISDLIST={Y N}] [,PMDLIST={Y N}] [,LISTDS={Y N}] [,LINCR=(first line number, increment)]

OPERATION	OPERANDS
AT	instruction location [, . . .]

OPERATION	OPERANDS
BACK	DSNAME=data set name

OPERATION	OPERANDS
BEGIN	application name [application parameters]
Note: For MT/T use only	

OPERATION	OPERANDS
BRANCH	INSTLOC=instruction location

OPERATION	OPERANDS
BUILTIN	NAME=command name, [,EXTNAME=BPKD macro instruction name] [,DSNAME = data set name]

OPERATION	OPERANDS
C	

OPERATION	OPERANDS
CA	

OPERATION	OPERANDS
CALL	[NAME= { entry-point name } { module name } [,module parameters]]

OPERATION	OPERANDS
CANCEL	BSN=batch sequence number

Form 1

OPERATION	OPERANDS
CATALOG	DSNAME=current data set name [,STATE={N U}] [,ACC={R U}] [,NEWNAME=new data set name]

Form 2

OPERATION	OPERANDS
CATALOG	GDG=generation data group name, GNO=number of generations [,ACTION={A O}] [,ERASE={Y N}]

OPERATION	OPERANDS
CB	

OPERATION	OPERANDS
CDD	DSNAME=data set name [, { data definition name { (data definition name, . . .) } }]

OPERATION	OPERANDS
CDS	DSNAME1=input data set name [(member name [, . . .])] ,DSNAME2=Copy data set name [(member name)] [,ERASE={Y N}] [, {COPYBASE=first line number [COPYINCR=increment]} {REPLACE={R I}}]

OPERATION	OPERANDS
CLOSE	[DSNAME=data set name] [,TYPE=T] [,DDNAME=data definition name]

OPERATION	OPERANDS
CHGPASS	[password]

OPERATION	OPERANDS
CONTEXT	[N1=starting position] [,N2=ending position], STRING1=search string [,STRING2=replacement string]

OPERATION	OPERANDS
CORRECT	[N1=starting line] [,N2=ending line] [,SCOL=first column] [,CORMARK=correction markers] [,CHAR={C M H}]

OPERATION	OPERANDS					
DATA	DSNAME=data set name [,RTYPE= { <table style="display: inline-table; vertical-align: middle;"> <tr><td>FTN</td></tr> <tr><td>CARD</td></tr> <tr><td>S</td></tr> <tr><td>{I</td></tr> <tr><td>{LINE}</td></tr> </table> } [,BASE=first line number, INCR=increment]]	FTN	CARD	S	{I	{LINE}
FTN						
CARD						
S						
{I						
{LINE}						

OPERATION	OPERANDS
DDEF	DDNAME=data definition name [,DSORG={VI VS VP}] ,DSNAME=data set name

OPERATION	OPERANDS
DDNAME?	[JOB LIB={Y N}]

OPERATION	OPERANDS
DEFAULT	{operand=[value]} [, . . .]

OPERATION	OPERANDS
DELETE	[DSNAME=data set name]

OPERATION	OPERANDS
DISABLE	
OPERATION	OPERANDS
DISPLAY	{data field or expression [, . . .]} {ID? data field name }
OPERATION	OPERANDS
DMPRST	FROMDEV={2311 2314 2400}, FRVOLID={valid (valid [,valid])}, TODEV={2311 2311 2400} [,TOVOLID=valid (valid[,valid] PRIVATE)] [,NEWVLID=valid] [,WRITCHK={YES NO}] [,LABEL={RETAIN NO}] [,RUNMODE={BACK FORE}]
OPERATION	OPERANDS
DSS?	[NAMES= {data set name {(data set name, . . .)}]
OPERATION	OPERANDS
DUMP	{data field or expression [, . . .]} {ID? data field name }
OPERATION	OPERANDS
EDIT	DSNAME=data set name [(member name)] [, RNAME=region name] [,REGSIZE=region name length]
OPERATION	OPERANDS
ENABLE	
OPERATION	OPERANDS
END	
OPERATION	OPERANDS
ERASE	[DSNAME=data set name]
OPERATION	OPERANDS
EVV	DEVICE=device type, VOLUME=(volume serial number[, . . .])
OPERATION	OPERANDS
EXCERPT	DSNAME=data set name [(member name)] [,RNAME=region name] [,N1=starting line [,N2=ending line]]

OPERATION	OPERANDS
EXCISE	[N1=starting line] [,N2=ending line]

OPERATION	OPERANDS
EXECUTE	DSNAME=data set name

OPERATION	OPERANDS
EXHIBIT	OPTION = { BWQ [, TYPE={ALL BSN-number}] { UID [, TYPE={ALL CONV BACK UID userid}] } }

OPERATION	OPERANDS
EXPLAIN	$\left[\left\{ \begin{array}{l} \text{MSGID} \\ \text{ORIGIN} \\ \left\{ \begin{array}{l} \text{word} \\ \text{TEXT} \\ \text{RESPONSE} \\ \text{MSGE} \\ \text{MSGS} \end{array} \right\} \end{array} \right\} \left[\text{,message identification} \right] \right]$

OPERATION	OPERANDS
FTN	NAME=module name [,STORED= { Y } { N }] [,VERID=version identification] [,ISD={Y N}] [,SLIST={Y N}] [,OBLIST={Y N}] [,CRLIST={Y N}] [,STEDIT={Y N}] [,MMAP={Y N}] [,BCD={Y N}] [,PUBLIC={Y N}] [,LISTDS={Y N}] [,LINCR=(first line number, increment)]

OPERATION	OPERANDS
GO	

OPERATION	OPERANDS
IF	condition

OPERATION	OPERANDS
INSERT	[N1=preceding line] [,INCR=increment]

OPERATION	OPERANDS
JOBLIBS	DDNAME=data definition name

OPERATION	OPERANDS
K	

OPERATION	OPERANDS
KA	

OPERATION	OPERANDS
KB	

OPERATION	OPERANDS
KEYWORD	[COMNAME=command name]

OPERATION	OPERANDS
LINE?	DSNAME=data set name [{ line number (first line number, last line number) } [, ...]]

OPERATION	OPERANDS
LIST	[N1={starting position CLP LAST}] [,N2={ending position LAST}] [,CHAR={C H M}]

OPERATION	OPERANDS
LNK	NAME=module name [,STORED={Y N}] [,LIB=data definition name of library] [,VERID=version identification] [,ISD={Y N}] [,PMDLIST={Y N}] [,LISTDS={Y N}] [,LINCR=(first line number, increment)]

OPERATION	OPERANDS
LOAD	[NAME=entry-point name]

OPERATION	OPERANDS
LOCATE	[N1=starting position] [,N2=ending position] [,STRING=character string]

OPERATION	OPERANDS
LOGOFF	

OPERATION	OPERANDS
LOGON	user identification, [password] , [addressing] , [charge number] , [control section packing] , [maximum auxiliary storage] , [pristine], [user IVM]

OPERATION	OPERANDS
MCAST	[ECB=end-of-block character] [,CONT=continuation character] [,CLP=break character] [,TRP=transient statement prefix character] [,RCC=concatenation character] [,SSM=system scope mask] [,USM=user scope mask] [,KC=keyboard/card reader character] [,RS=carriage return supression character] [,CP=command-prompt string]

OPERATION	OPERANDS
MCASTAB	[INTRAN={Y N}] [,OUTRAN={Y N}]

OPERATION	OPERANDS
MODIFY	SETNAME=data set name [,CONF=R] [,LRECL=record length, KEYLEN=keylength, RKP=key displacement, RECFM={V F}] [,FTN={Y N}]

OPERATION	OPERANDS
NUMBER	[N1=starting line] [,N2=ending line] [,NBASE=base number] [,INCR=increment]

OPERATION	OPERANDS
PC?	[NAMES= { (data set name { (data set name, ...) })]

OPERATION	OPERANDS
PERMIT	DSNAME={data set name °ALL} [,USERID={{user identification[, . . .]} °ALL}] [,ACCESS={R RO RW U}]

OPERATION	OPERANDS
PLI	[NAME=module name] [,PLIOPT=compiler option list] [,PLCOPT=language controller options] [,SOURCEDS=source data set name] [,MERGELST=converter input list] [,MERGEDS=converter input data set] [,MACRODS=intermediate data set name] [,PRVDS=data set name]

OPERATION	OPERANDS
POD?	[PODNAME=data set name] [,DATA=Y] [,ALIAS=Y] [,MODULE= {name} {ALL}]

OPERATION	OPERANDS
POST	

OPERATION	OPERANDS
PRINT	DSNAME=data set name [,STARTNO=first byte position] [,ENDNO=last byte position] [,PRTSP= { EDIT }] [{ 1 }] [{ 2 } [,HEADER=H] [,LINES=lines per page] [,PAGE=P]] [{ 3 }] [,ERASE={Y N}] [ERROPT={ACCEPT SKIP END}] [,FORM=paper form] [,STATION=station identification]

OPERATION	OPERANDS
PRMPT	MSGID=message identification [{INSERTn=inserted characters} [, . . .]]

OPERATION	OPERANDS
PROCDEF	NAME=procedure name [,DSNAME=data set name]

OPERATION	OPERANDS
PROFILE	CSW={N Y}

OPERATION	OPERANDS
PUNCH	DSNAME=data set name [,STARTNO=first byte position] [,ENDNO=last byte position] [,STACK={1 2 3 EDIT}] [,ERASE={Y N}] [,FORM=card stock]

OPERATION	OPERANDS
QUALIFY	MNAME=[link-edited module name.] object-module name

OPERATION	OPERANDS
REGION	[RNAME=region name]

OPERATION	OPERANDS
RELEASE	DDNAME=data definition name [,DSNAME=data set name] [,{SCRATCH HOLD}]

OPERATION	OPERANDS
REMOVE	{statement number [, . . .]} {ALL}

OPERATION	OPERANDS
RET	DSNAME=data set name ,RET={P T} {L C} {U R}

OPERATION	OPERANDS
REVISE	[N1=starting line] [,N2=ending line] [,INCR=increment]

OPERATION	OPERANDS
RUN	[LOC=entry point name]

OPERATION	OPERANDS
SECURE	{(TA=number of devices, [type of device])} {(DA=number of devices, [type of device])} [, ...]

OPERATION	OPERANDS
SET	{data location=value} [, . . .]

OPERATION	OPERANDS
SHARE	DSNAME=data set name, USERID=owner's user identification] [,OWNERDS={owner's data set name}*ALL,}]

OPERATION	OPERANDS
STET	

OPERATION	OPERANDS
STOP	

OPERATION	OPERANDS
SYNONYM	{term=[value]} [, . . .]

OPERATION	OPERANDS
TIME	[MINS=minutes]

OPERATION	OPERANDS
TV	DSNAME1=tape data set name [,DSNAME2=VAM data set name]

OPERATION	OPERANDS
UNLOAD	[NAME=entry-point name]

OPERATION	OPERANDS
UPDATE	

OPERATION	OPERANDS
USAGE	

OPERATION	OPERANDS
VT	DSNAME1=VAM data set name [,DSNAME2=tape data set name]

OPERATION	OPERANDS
VV	DSNAME1=current data set name [,DSNAME2=new data set name]

OPERATION	OPERANDS
WT	DSNAME=current data set name [,DSNAME2=tape data set name] [,VOLUME=tape volume number] [,FACTOR=blocking factor] [,STARTNO=first byte position] [,ENDNO=last byte position] [,PRTSP={1 2 3 EDIT}] [,HEADER=H] [,LINES=lines per page] [,PAGE=P] [,ERASE={Y N}]

OPERATION	OPERANDS
ZLOGON	

Appendix H: Carriage and Punch Controls

The carriage and punch controls shown in Tables 15 and 16 are recommended as standard; four of them are standard FORTRAN control characters. *They are installation variable, however, depending upon system output routines and, for carriage control, the printer's carriage control tape.

In conversational mode where `sysout` is the user's terminal rather than an offline printer, all carriage control characters other than 0 and 1 cause a single line skip to occur prior to printing of the line. The carriage control character 0 causes an additional line skip prior to printing of the line, as with offline printer processing. The carriage control character 1 causes three lines to be skipped prior to printing of the line.

Table 15. Carriage Control Characters

FUNCTION	CHARACTER
*Skip no line before printing	+
*Skip 1 line before printing	blank
*Skip 2 lines before printing	0
Skip 3 lines before printing	-
*Skip to channel 1 before printing	1
Skip to channel 2 before printing	2
Skip to channel 3 before printing	3
Skip to channel 4 before printing	4
Skip to channel 5 before printing	5
Skip to channel 6 before printing	6
Skip to channel 7 before printing	7
Skip to channel 8 before printing	8
Skip to channel 9 before printing	9
Skip to channel 10 before printing	A
Skip to channel 11 before printing	B
Skip to channel 12 before printing	C

*Standard FORTRAN control characters.

Table 16. Punch Control Characters

FUNCTION	CHARACTER
Select punch pocket 1	V
Select punch pocket 2	W

*As used in this book, "FORTRAN control characters" refers to the control characters defined by American National Standard FORTRAN, ANSI X3.9-1966.

Part One — Nonconversational

In part one of this appendix, a user enters a program, CONV, to read matrix A and matrix B as input. The product, matrix C, is then computed and all three matrixes are printed as output. The matrices are real and stored a column at a time (i.e., the leftmost subscript varying most rapidly) in order to minimize the amount of storage "paging" required. The card listing for CONV is shown in Figure 25. The program compiles without error and is executed.

After the compilation has been completed, the user receives a compilation listing, shown in Figure 26, and a SYSOUT listing, shown in Figure 27, which includes the commands, system messages, and three matrices A, B, and C.

```

LOGON ADUSERID,ADACCT29
FTN NAME = CONV, ISD = N

C      MATRIX MULTIPLICATION EXAMPLE
      DIMENSION A(8,8),B(8,8),C(8,8)
      DATA C/64*C.0/
      READ(2,1) L,M,N
      DO 10 J=1,M
10     READ(2,2) (A(J,I),I=1,L)
      DO 20 J=1,N
20     READ(2,2) (B(J,I),I=1,M)
      DO 30 K=1,M
      DO 30 J=1,N
      DO 30 I=1,L
30     C(I,J)=C(I,J)+A(I,K)*B(K,J)
      DO 40 J=1,M
40     WRITE(3,3) (A(J,I),I=1,L)
      WRITE(3,4)
      DO 50 J=1,N
50     WRITE(3,3) (B(J,I),I=1,M)
      WRITE(3,5)
      DO 60 J=1,N
60     WRITE(3,3) (C(J,I),I=1,L)
      STOP
1      FORMAT (3I2)
2      FORMAT (8F10.0)
3      FORMAT (' ',8F10.3)
4      FORMAT (14X,'X')
5      FORMAT (14X,'=')
      END

      RUN CONV
      3 3 3
          3.0      11.0      19.0
          5.0      13.0      23.0
          7.0      17.0      29.0
          31.0     43.0      59.0
          37.0     47.0      61.0
          41.0     53.0      67.0
%END
LOGOFF

```

Figure 25. Card Listing for CONV

VERSION 07/10/67 OF THE TSS FORTRAN COMPILER ENTERED

PAGE 001

THE MODULE NAME AND VERSION FOR THIS COMPILATION ARE CONV ,V5

OPTIONS--PUBLIC CSECT(N),BCD MODE(N),PRODUCE ISD(N), LISTINGS--SOURCE(Y),OBJECT(N),CROSS REF(N),SYMBOL TABLE(N),MEMORY MAP(N).

CONV ,V5

PAGE 002

```

100 C      MATRIX MULTIPLICATION EXAMPLE
200      DIMENSION A(8,8),B(8,8),C(8,8)
300      DATA C/64*0.0/
400      READ(2,1) L,M,N
500      DO 10 J=1,M
600 10     READ(2,2) (A(J,I),I=1,L)
700      DO 20 J=1,N
800 20     READ(2,2) (B(J,I),I=1,M)
900      DO 30 K=1,M
1000     DO 30 J=1,N
1100     DO 30 I=1,L
1200 30     C(I,J)=C(I,J)+A(I,K)*B(K,J)
1300     DO 40 J=1,M
1400 40     WRITE(3,3) (A(J,I),I=1,L)
1500     WRITE (3,4)
1600     DO 50 J=1,N
1700 50     WRITE(3,3) (B(J,I),I=1,M)
1800     WRITE (3,5)
1900     DC 60 J=1,N
2000 60     WRITE(3,3) (C(J,I),I=1,L)
2100     STOP
2200 1     FORMAT (3I2)
2300 2     FORMAT (8F10.0)
2400 3     FORMAT (' ',8F10.3)
2500 4     FORMAT (14X,'X')
2600 5     FORMAT (14X,'*')
2700     END

```

CONV ,V5

PAGE 003

CONV SIZE 2204 BYTES

ENTRY NAME	LOC HEX
CONV	00000000

EXTERNAL REFERENCES

CHCB01	CHCIA1	CHCIE1	CHCIU1	CHCIN2
CONV#C	SIZE	908 BYTES	07/13/67 14:23:15	

CODE	LOC HEX	SIZE	BYTES
	00000000		864 BYTES

NUMERIC CONSTANTS	LOC HEX	SIZE	BYTES
	00000360		44 BYTES

CONV ,V5

PAGE 004

CONV#P SIZE 1296 BYTES

07/13/67 14:23:15

REGISTER SAVE AREA	LOC HEX	SIZE	BYTES
	00000000		76 BYTES

CONVERSION CONSTANTS	LOC HEX	SIZE	BYTES
	0000004C		24 BYTES

ADDRESS CONSTANTS	LOC HEX	SIZE	BYTES
	00000064		124 BYTES

ALPHAMERIC	LOC HEX	SIZE	BYTES
	000001F0		43 BYTES

LOCAL TEMPORARY STORAGE	LOC HEX	SIZE	BYTES
	000004F8		24 BYTES

NON-COMMON VARIABLES (TOTAL)	LOC HEX	SIZE	BYTES
	000001E0		792 BYTES

CONV ,V5

PAGE 005

COMPILATION COMPLETED

Figure 26. Compilation Listing for CONV

```

LOGON ADUSERID,ADACCT29
FTN NAME=CONV

CONV
  3.000    11.000    19.000
  5.000    13.000    23.000
  7.000    17.000    29.000
      X
 31.000    43.000    59.000
 37.000    47.000    61.000
 41.000    53.000    67.000
      =
1279.000  1653.000  2121.000
1579.000  2045.000  2629.000
2035.000  2637.000  3393.000
LOGOFF

```

Figure 27. SYSOUT Listing for CONV

Part Two — Conversational

In this part, a user writes and compiles a program to solve the same problem as solved in part one. He then runs it on-line. He uses his console for both input and

output, switching it to become the equivalent of `sysin` and `sysout`, i.e., the standard system input and output, as shown in Figure 28. A sample `sysout` listing is shown in part one.

```

USR:  (presses attention button or dials up system)
      LOGON ADUSERID,MYPASS*, ,ADACCT29
SYS:  BOO1 LOGON TASKID=FFF9 07/25/68
S,U:  FTN NAME=CONV
S,U:  0000100C    MATRIX MULTIPLICATION EXAMPLE
S,U:  0000200    DIMENSION A(8,8),B(8,8),C(8,8)
S,U:  0000300    DATA C/64*0.0/
S,U:  0000400    READ(2,1) L,M,N
S,U:  0000500    DO 10 J=1,M
S,U:  0000600 10  READ(2,2) (A(J,I),I=1,L)
S,U:  0000700    DO 20 J=1,N
S,U:  0000800 20  READ(2,2) (B(J,I),I=1,M)
S,U:  0000900    DO 30 K=1,M
S,U:  0001000    DO 30 J=1,N
S,U:  0001100    DO 30 I=1,L
S,U:  0001200 30  C(I,J)=C(I,J)+A(I,K)*B(K,J)
S,U:  0001300    DO 40 J=1,M
S,U:  0001400 40  WRITE(3,3) (A(J,I),I=1,L)
S,U:  0001500    WRITE (3,4)
S,U:  0001600    DO 50 J=1,N
S,U:  0001700 50  WRITE(3,3) (B(J,I),I=1,M)
S,U:  0001800    WRITE (3,5)
S,U:  0001900    DO 60 J=1,N
S,U:  0002000 60  WRITE(3,3) (C(J,I),I=1,L)
S,U:  0002100    STOP
S,U:  0002200 1   FORMAT (3I2)
S,U:  0002300 2   FORMAT (8F10.0)
S,U:  0002400 3   FORMAT (' ',8F10.3)
S,U:  0002500 4   FORMAT (14X,'X')
S,U:  0002600 5   FORMAT (14X,'=')
S,U:  0002700    END
SYS:  The system asks whether it should continue processing.

```

Figure 28. Conversational SYSIN-SYSOUT for CONV

```

USR:      Y
SYS:      The system informs the user that it found no errors.
S,U:      PRINT LIST.CONV(0) , , , EDIT, ERASE
S,U:      CALL CONV
SYS:      The keyboard unlocks indicating the system will accept input data.
USR:      3 3 3
SYS:      The keyboard unlocks indicating the system will accept input data.
USR:      3.0      11.0      19.0
SYS:      The keyboard unlocks indicating the system will accept input data.
USR:      5.0      13.0      17.0
SYS:      The keyboard unlocks indicating the system will accept input data.
USR:      7.0      17.0      29.0
SYS:      The keyboard unlocks indicating the system will accept input data.
USR:      31.0     43.0      59.0
SYS:      The keyboard unlocks indicating the system will accept input data.
USR:      37.0     47.0      61.0
SYS:      The keyboard unlocks indicating the system will accept input data.
USR:      41.0     53.0      67.0
PGM:      3.000     11.000    19.000
          5.000     13.000    23.000
          7.000     17.000    29.000
          X
          31.000    43.000    59.000
          37.000    47.000    61.000
          41.000    53.000    67.000
          =
          1279.000  1653.000  2121.000
          1579.000  2045.000  2629.000
          2035.000  2637.000  3393.000
SYS:      CHCIW STOP
USR:      ERASE SOURCE.CONV
S,U:      LOGOFF
S,U:      The system acknowledges log-off.

```

Figure 28. Conversational SYSIN-SYSOUT for CONV (continued)

- % (equivalent to PCS statement counter) 98, 103
- %COM as a symbol 99
- %END 29, 44, 136
- %L 59, 60, 71, 75
- %END 130

- ABEND command 152
- ABENDREG command 152
- & COM as a symbol 99
- & END 130
- ASM command
 - assembler language programs 146, 152
- AT command
 - description 101, 102
 - example 38, 17
 - general form 152
- Attention button 10, 20, 52, 149-150
- Attention considerations 149-150

- BACK command
 - continuation following 13
 - data set requirements 146
 - example 14, 42
 - general form 152
- Backspace 133
- Backward pointer 120
- Base register adcon 87
- Basic DDEF command 134
- Basic sequential access method (BSAM) 124-125
- Batch sequence number 27
- BCD FTN command parameter 80, 82
- BEGIN command 152
- Blank character 130
- Blank COMMON 123
- BLKSIZE Parameter for DDEF command 140
- BLOCK DATA Subprogram
 - example 34-35
 - linkage editing 118
 - listing 85
 - name assignment 115-116
- Blocking records 126
- Braces {} 151
- Brackets [] 151
- BRANCH command
 - description 105
 - general form 153
- BSN 27
- BSAM 124-125
- BUFNO parameter for DDEF command 141
- BUILTIN command 153

- C command 153
- CA command 14, 58, 153
- CALL command
 - description 105
 - directory of examples 17
 - general form 153
- Call data definition command
(see CDD command)
- Calling
 - assembler language programs 119-123
 - DVCHK 106
 - FORTTRAN subroutines 121-122
 - OVERFL 106
- CANCEL command
 - example 14, 59
 - general form 153
- Canceling
 - data definitions 9
 - line 20, 49
 - PCS statements 103-104
- Card decks from outside TSS/360 132
- Carriage control characters 31, 140, 162
- CATALOG command
 - data set requirements 146
 - directory of examples 14
 - general form 153
 - shared data sets 114
- Catalog
 - access types 8-10
 - data sets 8-9
 - indexes 3, 8-9
 - prestored data 9
 - private data set 41
- CB command 14, 58, 153
- CDD command
 - data set requirements 146
 - example 15, 46
 - general form 153
- CDS command
 - data set requirements 146
 - description 12
 - example 15, 49
 - general form 154
- CF 102
- Changing
 - contents of variable 104-105
 - program flow 104-105
- Character constant in PCS expression 100
- Charge number 1
- CHGPASS command 154
- CLOSE command 154
- Code optimization 107
- Code file (CF) 102
- Command
 - descriptions 151
 - directory 14
 - executing in FORTRAN programs 117
 - formats 151
 - language 2
 - operands 151
 - parameter defaults 20
 - prompting 4
- Comment line 22
- COMMON
 - control section listing 85
 - data use 123
 - display 103
 - efficiency considerations 108
 - multiple execution 111
 - statement misuse 119
 - symbol resolution 111
 - unloading blocks 60
 - variable storage allocation 60
 - variable storage size 95
- Compilation
 - sample listing 164
 - terminal 22

Compile and run	5, 26	printing	30, 31
Compiler		protection	9
(see FORTRAN)		record	
Complex numbers	120	(see record)	
Constants in PCS expressions	100	reference number	27, 60
CONTEXT command	154	removing catalog entry	8
Continue		requirements for commands	146-147
character	78	sharing	9
form of CALL command	104	structure	124
compilation	23	VI	125, 6
line	20, 85	virtual index sequential	125, 6
Control character		virtual partitioned	125, 7
carriage	31, 162	virtual sequential	125, 6
Control section		VP	125, 7
display	102	VS	125, 6
listing	84	Data set definition	
name	83, 99	description	7, 146
packing	110	requirements for commands	146-147
public	115	Data set name	
shared	115	description	3, 138
Conversational mode		list	24, 123
processing	1	qualification	3
sample program	165	restrictions	135
source statement correction	77	source	35, 49
SYSIN/SYSOUT example	163	Data set organization	
task initiation	2, 20	requirements for commands	146-147
Copy data set command		table	131
(see CDS command)		Data set reference number	27, 60
Copying data sets	12	Data set status? command	
CORRECT command	16, 154	(see DSS? command)	
Correction		DCB	140-142, 52
DDEF command	145	DDEF command	
PCS statement errors	76, 77	cataloged data set	40
source statement	22, 76, 77	correction	145
Creation		ddname parameter	7, 137
generation data group	65	defaulting	136
job library	26	description	132
listing data set name	24	diagnostics	145
source data set name	49	directory of examples	14
CRLIST FTN command parameter	81	general form	154
CRL		interchangeability	7
(see cross-reference table)		multiple execution use	60, 111
Cross-reference listing		new data set	137
description	91	old data set	137
FTN command parameter	81	omission	8
sample	90	parameters	137
Cross-reference table	105	retrieving	8, 46
CSECT		storing (see CDD command)	
(see control section)		uncataloged data set	60
Data control block parameters	140-142, 52	ddname parameter of DDEF command	137
DATA command		DDNAME? command	154
data set requirements	146	Debugging	36, 96
directory of examples	16	Default compilation parameters	26
general form	154	DEFAULT command	18, 146, 154
interruption considerations	149	Default command parameters	20, 24
prompts	49	Default FTN command parameters	80-82
Data set		Default option listing	
cataloging	8, 135	description	83-85
creation	124	sample	84
erasing	12	Defective track	133
generation	130	Define data command	
indexed	49	(see DDEF command)	
labels	60, 139	Defining job library	26, 140
name, moving	61	DELETE command	
name, qualifiers	8	data set requirements	146
name, restrictions	135	example	63
organization	6, 124	general form	154
partitioned	125, 7	shared data set	114
physical sequential	7, 124, 125	use	15
		Deletion line during DATA command	50
		Delimiters for PCS statements	114

Determining size of data set	56	ERR option	132
DEVD parameter for DDEF command	141	ERROR parameter of PRINT command	31
Device reserving	9	Error	
Diagnostics		code	22, 79
(see messages)		I/O	132
Diagnostic messages		messages	
(see messages)		(see messages)	
Dialog with system	2	MODIFY command	28
Dial up system	20	nonconversational mode	45
DIR macro instruction	119	recovery procedure	27
DISABLE command	16, 155	EVV command	155
DISP parameter for DDEF command	32, 131, 135, 140	Exception handling	117, 119, 132
DISPLAY command		EXCERPT command	16, 155
description	102	EXCISE command	16, 156
example	36, 17	EXECUTE command	
general form	155	data set requirements	147
Displaying records	12	example	14, 42
Disposition of data sets at log-off	110	general form	156
DMPRST command	155	Executable statement	116
DO loop efficiency considerations	109	EXHIBIT command	156
ddname parameter of DDEF command		EXIT statement	132
description	133, 134	EXPLAIN command	156
dsname parameter of DDEF command	135, 138, 145	Explicit symbol qualification	99
dsorg parameter of DDEF command	135, 137	Exponent-overflow	105, 117
DSS? command		Exponent-underflow	105, 117
data set requirements	146	Expression File (EF)	93-94
example	15, 62	External Name List (ENL)	96
general form	155	External symbols	115, 116
Dummy arguments in PCS command	106		
DUMP command		F-code message	80
data set requirements	147	File sequence number	139
description	109	Fixed length record format	126, 127
example	17, 38, 103	Fixed-point divide	105
general form	155	Fixed-point overflow	105, 117
duplication of symbols	112, 113, 115	Floating continuation character	75
DVCHK	106	Floating-point computations	116
		Floating-point divide exception	105, 117
		FORMAT statement	14, 127
		format of commands	151
E-code message	79	Formatted records	127
EDIT command	15, 147, 155	FORMNO for PRINT command	31
EDIT feature of PRINT command	31	Forms of a program	56
efficiency (see optimization)		Forward pointer	120
ellipsis (...)	151	FORTRAN CALLS	121
ENABLE command	16, 155	FORTRAN	
END command	15, 147, 155	card format	22
END statement	23, 76	control characters	162
End option of READ statement	44	external symbols	99
END FILE statement	32, 133	diagnostic messages	78, 79, 92
End-data-set condition	132	error checking	23
End-of-data indicator	29	internal symbols	99
Entering		I/O statement	8
commands	4	listings	83-91
command sequence	13	optimization	107, 110
data	12	parameters	35, 80, 82
source statements for punching	78	command records	8
Entry point register	120	restrictions	93-96
ENTRY statement	115	statement numbers	99
EQUIVALENCE statement	108, 119	variables, initial content	118
ERASE		FORTRAN IV library subprograms	
PRINT command parameter	31	calling	118
PUNCH command parameter	56	description	2
ERASE command		list of names	92
data set requirements	147	substitution	97
directory of examples	15	Free form terminal entry	26
general form	155	FTN command	
shared data set	114	data set requirements	146
Erasing		directory of examples	17
library	63	general form	156
listing data set	58	parameters	80
member of partitioned data set	63	FTxxFyyy	7, 134
program library list	113		

Full messages	20	LINCR FTN command parameter	82
FUNCTION subprogram	5, 120	LINE? command	
GDC	65, 7	data set requirements	147
Generation data group	65, 7	directory of examples	16
Global corrections	76	general form	157
Global errors	24	Line data set	6
GO command	104	Line number FTN command parameter	82
description	17	Line number increment FTN command	
directory of examples	17	parameter	82
general format	156	Linkage	
HEADER parameter of PRINT command	31	between FORTRAN and assembler language	
Hexadecimal constant in PCS		programs	122
expression	100	type I	119
HOLD option on DDEF command	139	Linkage editor	
Housekeeping methods	62	efficiency considerations	109, 110
Hyphen (-) as continue character	78	program libraries	112, 113
IF command		use	117
description	103	LIST	
example	38	command	16, 157
general format	156	data set name	24, 116
Illegal address assignment	47	LISTDS operand of FTN command	13, 82
IMSK parameter for DDEF command	132	Listing data set	6, 24, 82, 83
Indexed data set	49	LNK command	157
Input/Output		LOAD command	
error messages	133	before PCS statements	38
list	55	BLOCK DATA subprogram	35
procedures	5	CALL statement	47
statement	6	data set requirements	147
tape	40	errors	98
INSERT command	16, 156	example	17, 38
Integer constant in PCS expression	100	general form	157
Integer result register	120	interruption considerations	150
Internal symbol dictionary		object modules	120
creation	81	program with errors	28
size	96	Load-and-run form of CALL command	104
use in PCS	97, 110	Local corrections	76
Internal symbols		LOCATE command	16, 157
effect of linkage editing	99	Logical record	
Interruptions		(see record)	
during message printout	149	LOGOFF command	
handling	52, 132, 149	conversational task	12
resuming execution	36, 150	directory of examples	14
symbolic location	98	general form	157
ISD		nonconversational task	13, 44
(see Internal symbol dictionary)		LOGON command	
ISD FTN command parameter	81	conversational task	2, 20
JOBLIB		directory of examples	14
(see Job Library)		general form	158
JOBLIB option in DDEF		nonconversational task	43
command	26, 140	Log-on process	2, 20
JOBLIBS command	156	L.RECL parameter in DDEF	
Job library		command	131, 140
adding	10	M-machine code control character	162
contents	10	MCAB command	158
creation	34, 112	MCAB command	158
use	10	MMAP FTN command parameter	81
Job processing	44	Magnetic tapes,	
Joining the system	1	non-TSS/360	132
K command	157	Master Index	3
KA command	14, 58, 157	Memory map FTN command	
KB command	14, 58, 157	parameter	81
KEYWORD command	157	Messages	
LABEL parameter in DDEF command	60, 146	conversational output	79
Library hierarchy	12	DDEF command	145
library		diagnostic	4, 19
obtaining information	63	option	20
search	12, 47, 48	prompting	19
user (see user library)		response	19
		types	19
		Metasymbol	151
		Misspelling during DATA command	49

Mixed arithmetic	109	Page numbering parameter of PRINT	31
Mixed input (card and keyboard)	74	command	109
Mixed mode	13	Page utilization	109
MODIFY command		Paging	
data set requirements	147	reduction	115
example	16, 28	Parameter area	120
general form	158	Parameter list	
Modification		indication in PSECT listing	87, 88
data sets	12	length	120
source statement	28	register	120
terminating	28	Partitioned data set	
Module name FTN command parameter	80	(see virtual partitioned)	
Multiple executions	111	Password	1, 20
NAME FTN command parameter	80	PAUSE statement	
NAMELIST		description	6
indication in PSECT listing	87	difference between STOP and RETURN	117
size	103	PCS	
writing records	6, 129	(see program control system)	
END	130	PCSOUT data set	98, 102
Names		PC? command	158
list data set	24, 116	% (equivalent to PCS statement counter)	98, 103
qualification	113	% END	29, 44, 136
rules	115	% L	59, 60, 71, 75
shared data set	55	PERMIT command	
source data set	35	catalog alteration	54
Non-COMMON variable storage size	95	data set requirements	147
Nonconversational		description	10
LOGON command	43	example	15, 54
processing	20, 44, 60	general form	158
program control system	98	Permitting access	54
sample program	163-164	Phase 2 internal table	96
SECURE requirements	143	PF	
task initiation	20, 42	(see program file)	
Nonexecutable statement	99	Physical sequential	
NUMBER command	16, 158	data set	125
Number of devices required	133	organization	7
Numerical constants indication		record format	129, 131
in CSECT listing	87	PLI command	158
Object listing FTN command parameter	81	POD? command	15, 147, 158
Object module	5, 22, 119	Positioning statement	133
Object program module		POST command	158
(see object module)		Preset data table (PDT)	95
Object time efficiency	107	Prestore	
OBLIST FTN command parameter	81	DDEF commands	8
OFF		procedure	13
(see LOGOFF command)		source program	28, 49
Operand		Prestored source data set FORTRAN	
keyword	151	parameter	81
positional	151	PRINT command	
OPTCD parameter for DDEF		data set record formats	31
command	141	data set requirements	147
Optimization		description	12
compiler	107	directory of examples	17
page reference	93	general form	159
unloading	110	Priority	1
user program	107, 108	PRIVATE option of	
Optimization table		DDEF command	139
size	96	Private volume	
OPTION parameter of DDEF		device allocation	9
command	140	job library	112
Order of search	112	mounting	9
Output		Privilege class	1
destination	95	PRMPT command	159
use on other system	131	Problem program	
Output module listing	83, 84, 86, 89	communication	5
OVERFL	106	residence	10
PAD parameter of DDEF		PROCDEF command	14, 147, 159
command	141	PROFILE command	18, 147, 159
		Program	
		interruption	38, 105, 117
		library	111

manipulation	56	fixed-length records	126, 127
module (see object module)		END option	133
Program control system		restrictions	137
command language	98	tape devices	6
data set printing	39	terminal	5
diagnostics	105	variable length records	126
dynamic statement	26, 98	Read tape command	
expressions	100	(see RT command)	
immediate statement	36, 97	Read-and-write shared data set	
list of commands	17, 97	access	10, 113
optimization considerations	110	Read-only	
removal of alterations	37	cataloged data set access	10
restrictions	110	shared data set access	10, 113
statement counter	98, 103	Real constant in PCS expression	100
statement number	38	RECFM parameter in DDEF	
use	13, 97	command	140
%	98, 103	Record	
Program data		format	124
end indication	44	length	126
Program File (PF)	95	structure	131
Program library	111	Reference	
Program library list		indication on listing	85
description	10, 26, 111, 112	resolving	47
order of search	114	to subroutine	47
removal of library	113	REGION command	16, 147, 159
Program module		Register	
(see object module)		save area	87
Program module dictionary size	96	use	120
Program representation file (PRF)	95	Relative generation name	65
Program status word	36	Relative reference	65
Programming considerations	107	RELEASE command	
Prompting	20, 24	data set requirements	147
Protection		description	16
against unauthorized use	1	example	15, 39, 52
storage	2	general form	159
Prototype section		interruption considerations	52
listing	84	job library	10
name	116	multiple execution	60, 111
PS		REMOVE command	
(see physical sequential)		description	103
PSECT		example	17, 39
(see prototype section)		general form	159
PSW	36	REPEAT command	158
PUBLIC FTN command parameter	27, 82	Replacing records	12
Public		Reserving devices	13
considerations	115	Resolving external references	47
control section attribute	25	Resuming execution after interruption	36, 44, 150
volumes	9	RET command	159
PUNCH command		Retaining data sets	9, 139
control characters	162	RETPD parameter of DDEF	
data set requirements	147	command	139
example	17, 56	Retrieving DDEF commands	9, 46
general form	159	RETURN macro instruction	120
prestored source statements	78	Return register	120
Qualification		Reviewing source statements	28
data set name	3	REVISE command	160
internal symbols	99	REWIND statement	133
names	55	RT command	12
QUALIFY command		RUN command	
description	101	description	27, 104
example	17, 36, 38	general form	160
general form	159	Sample program	
R-value		conversational	165-166
READ statement		nonconversational	163
data sets	32, 132	Save area	120
defaulted data set	133	Save area register	120
DDEF command	27	Search order	10
direct-access device	5	SECURE command	
		description	9, 133
		example	14, 41, 60

general form	160	STORED FTN command parameter	77
Service routines	1	Storing DDEF commands	46
Session	2	Stroke (1)	99
SET command	104	Subprogram	5, 120
description	17, 38	FUNCTION	123
example	160	IBM-supplied	122
general form	105	name as CALL argument	120
restrictions	49	reference	5
Setting tab stops	79	SUBROUTINE	5
Severity code for compiler	79	Subroutine calls indicated in	85, 87
diagnostics	79	CSECT listing	5
SHARE command	55	SUBROUTINE subprogram	100
catalog	114, 144	Subscripted symbols	1
data set requirements	15, 54	Supervisor program	58
example	160	Switching between card and keyboard	81
general form	160	input	81
Shared data set	10, 54, 113	Symbol table edit FTN command parameter	89-90
access	10	Symbol table	102
authorization	115	listing	18, 147, 160
coding considerations	10	size	22
description	54	SYNONYM command	22
example	113	Syntactical errors	System
libraries	8	administrator	1
removal of catalog entry	117	catalog	3
Significance exception	119	library	10, 112
SIR macro instruction	81	manager	1
SLIST FTN command parameter	49	resources	13
SOURCE data set name qualifier	49	System routine residence	3
Source listing	82	SYSIN	2, 163
FTN parameter	83	conversational	2
sample	94	description	12-13
Source program	83	nonconversational	10, 112
complex restrictions	93	SYSLIB	117
listing	93	SYSOBF	SYSOUT
simple restrictions	93	conversational	163
Source statement	139	description	2
(see statement)	31	nonconversational	12-13
SPACE parameter of DDEF command	119	T option in RETURN macro instruction	120
Spacing option of PRINT command	117, 119	Tab stops	26, 74
SPEC macro instruction	74	Table of Initialized Variables	89, 90
Specification exception	76	Task	1, 2, 12, 20
Statement	74	Terminal	50
comment	76	card reader input	5-6
committed	74	I/O	50
continuation	74	keyboard input	58
END	74, 78	mode	1
format	74	operation	2
initial	74, 75	session	130
keyboard	74	Termination	4, 45
limitations	22, 24	input line	14, 160
line number	75	session	3
number	76	TIME command	3
partial	76	Time interval	3
tentative	95	Time Slice	81
Statement function expression file	81	Time Stamp	120
STEDIT FTN command parameter	160	Track facility	Triad table
STET command	5, 105	(see Optimization table)	23, 147, 160
description	17, 36	TV command	119
example	160	Type I linkage	125
general form	149	Unblocked records	Uncataloged data sets
interruption considerations	117	erasure at log-off time	23
PAUSE comparison	117	procedure	8
RETURN comparison	98, 99	Undefined	126
Storage map listing	108	record format (U)	47
Storage	3	references	
optimization	3		
protection	84, 87		
Storage references indication in	93		
CSECT listing			
Storage specification list (SPL)			

Underscore	150	Virtual partitioned	
Unformatted records		data set	7, 125
description	6	displaying characteristics	63
writing	130	organization	63
UNIT parameter of DDEF command	133, 138	Virtual sequential	
Unlimited access		creation	50
cataloged data sets	6	data set	6, 125
shared data sets	9-10	organization	6
UNLOAD command		record formats	126, 127
after interruption	52	Virtual storage	
COMMON blocks	60	description	2-3
efficiency considerations	110	structure	109
example	17, 47	VISAM	
FORTRAN compiler	35	(see virtual index sequential)	
general form	160	Volsqno	
log-off time	39	(see volume sequence number)	
object module	113	Volserno	
unreferenced programs	47	(see volume serial number)	
Unlocking keyboard	27	Volumes	3, 5
Unresolved references	114	Volume sequence number	139
UPDATE command	161	Volume serial number	61, 139
USAGE command	161	VOLUME parameter of DDEF command	139
User catalog	8	VP	
User defined library	10	(see virtual partitioned)	
User identification	1	VPAM	
User Library		(see virtual partitioned)	
cataloging	23	VS	
contents	10	(see virtual sequential)	
description	8, 17	VSAM	
organization	10	(see virtual sequential)	
Userid	1	VT command	15, 148, 161
USERLIB		VV command	15, 148, 161
(see User Library)			
V-value	122	W-Code message	79
VAM		Withdrawing authorization to shared	
(see virtual access method)		data sets	10, 54
Variables in array	123	Word boundaries exception	119
Variable-length parameter list	120	Write statement	
Variable-length record format (V)	127	cataloged data set	32
VERID FTN command parameter	81	DDEF command	27
Version identification	24, 81	description	30
Vertical stroke 	151	fixed-length records	126
VI		variable-length records	126
(see virtual index sequential)		Write tape command	
Virtual access method (VAM)	124	(see WT command)	
(see also: virtual index sequential;		WT command	
virtual partitioned; virtual sequential)		data set requirements	148
Virtual index sequential		example	17, 63
data control block parameters	141	general form	159
data set	6, 125		
organization	6	ZLOGON	14, 161
record formats	126		

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM Technical Newsletter

File Number S360-25
Base Publication No. GC28-2025-4
This Newsletter No. GN28-3204
Date February 1, 1972
Previous Newsletters None

IBM System/360 Time Sharing System: FORTRAN Programmer's Guide

© IBM Corp. 1967, 1968, 1970, 1971

This Technical Newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

79-80
117-118

A change to the text is indicated by a vertical line to the left of the change.

Summary of Amendments

- The description of the effect of a level-2, severity code "E" error diagnostic has been modified.
- Errors in the description of the parameters required by SYSOBF have been corrected.

