



**Systems Reference Library**

## **IBM System/360 Operating System:**

### **Time Sharing Option Guide**

**OS Release 21**

This publication describes the concepts, features and implementation of TSO, a general purpose time-sharing facility operating under the MVT configuration of the control program. This manual is intended for those who design, generate, and maintain a TSO installation. Topics discussed are:

- The capabilities and advantages of time sharing in general and TSO in particular.
- The programming languages and system facilities available to a TSO terminal user.
- The system configurations TSO requires.
- How to generate and maintain a TSO system.
- The program Products available with TSO.

Information in this publication for TSO with the Model 65 Multiprocessing (M65MP) configuration is for planning purposes until that item is supported.

The prerequisite publication is:

**IBM System/360 Operating System: MVT Guide, GC28-6720.**



### **Sixth Edition (July, 1972)**

This is a reprint of GC28-6698-4 incorporating changes released in the following Technical Newsletter:

GN28-2519 (dated April 15, 1972)

This edition applies to release 21 as updated by component release 360S-OS-586, of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM Systems, consult the latest **IBM System/360 and System/370 Bibliography**, Order No. GA22-6822, and the current **SRL Newsletter**, Order No. GN20-0360, for the editions 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 for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, N.Y. 12602. Comments become the property of IBM.

## Preface

This publication describes the concepts, features and implementation of TSO, a general purpose time-sharing facility operating under the MVT configuration of the control program. This manual is intended for those who design, generate, and maintain a TSO installation.

This publication discusses:

- The general capabilities and advantages of TSO.
- The command language, programming languages and system facilities available to a TSO terminal user.
- The system elements added to MVT for TSO.
- The generation and maintenance of a system with TSO.

There are four appendixes:

- A list of the TSO commands by function.

- A list of the IBM Program Products available for TSO users, and references to further documentation for them.
- A list of the Time Sharing Driver Entry Codes.
- A list of terminal messages requiring installation action.

There is a glossary of terms, a bibliography of related reading, and an index.

The prerequisite publication is:

**IBM System/360 Operating System: MVT Guide, GC28-6720.**

**Note:** For planning purposes, this manual includes information for components and capabilities that have been announced for availability after the delivery date for the Time Sharing Option. See your local IBM representative for the availability date for:

- TSO with Model 65 Multiprocessing.



# Contents

|   |    |   |     |
|---|----|---|-----|
| <b>Summary of Amendments</b> . . . . .                    | 7  | Compiling a FORTRAN Program . . . . .   | 41  |
| <b>Introduction</b> . . . . .                             | 9  | Testing FORTRAN Programs . . . . .  | 42  |
| Advantages of a Time Sharing System . . . . .             | 10 | PL/I . . . . .  | 43  |
| Using a Terminal . . . . .                                | 11 | Entering a PL/I Program . . . . .   | 44  |
| Starting and Stopping a Terminal Session . . . . .        | 11 | Compiling a PL/I Program . . . . .  | 44  |
| Working at the Terminal . . . . .                         | 12 | Program Execution . . . . .   | 44  |
| System Configuration . . . . .                            | 13 | Assembler Language . . . . .  | 44  |
| Terminals . . . . .                                       | 13 | Assembling the Program . . . . .  | 44  |
| Transmission Control Unit . . . . .                       | 14 | Test Mode . . . . .   | 45  |
| Swap Data Set Devices . . . . .                           | 14 | Other Compilers . . . . .   | 45  |
| The Relationship of TSO to the Operating System . . . . . | 15 | A Compiler Command Procedure . . . . .  | 46  |
| Execution of Background Jobs from the Terminal . . . . .  | 16 | Nested Procedures . . . . .   | 47  |
| Foreground-Background Compatibility . . . . .             | 16 | <b>Problem Solving</b> . . . . .  | 49  |
| Restrictions and Limitations . . . . .                    | 16 | ITF: BASIC . . . . .  | 49  |
| System Control . . . . .                                  | 17 | ITF: PL/I . . . . .   | 51  |
| Job Definition and Scheduling . . . . .                   | 17 | Code and Go FORTRAN . . . . .   | 52  |
| Tuning the Time Sharing System . . . . .                  | 17 | <b>System Summary</b> . . . . .   | 55  |
| Monitoring System Use and Performance . . . . .           | 18 | The Time Sharing Driver . . . . .   | 56  |
| System Security . . . . .                                 | 19 | Control Routines . . . . .  | 57  |
| User Verification . . . . .                               | 19 | The Time Sharing Control Task . . . . .   | 57  |
| Program Protection . . . . .                              | 20 | The Region Control Task . . . . .   | 59  |
| Data Set Security . . . . .                               | 20 | LOGON/LOGOFF . . . . .  | 60  |
| Authorizations . . . . .                                  | 20 | The Terminal Monitor Program . . . . .  | 61  |
| Capabilities of the TSO Command Language . . . . .        | 20 | TEST . . . . .  | 62  |
| IBM Program Products . . . . .                            | 22 | Service Routines . . . . .  | 63  |
| Problem-Solving . . . . .                                 | 22 | Command Processors and User Programs . . . . .                                      | 64  |
| Programming . . . . .                                     | 22 | Terminal I/O . . . . .  | 65  |
| Text and Data Handling . . . . .                          | 23 | The Message Control Program . . . . .   | 65  |
| <b>Command Language Facilities</b> . . . . .              | 24 | Mixed Environment MCPs . . . . .  | 65  |
| Conventions at the Terminal . . . . .                     | 24 | Terminal Interfaces . . . . .   | 65  |
| Logging On . . . . .                                      | 24 | Multi-Terminal Message Processors . . . . .   | 66  |
| Input Editing . . . . .                                   | 25 | Overview and Storage Map . . . . .  | 67  |
| Entry Modes . . . . .                                     | 26 | Time Sharing Algorithms . . . . .   | 68  |
| The Attention Key . . . . .                               | 26 | Time Slices . . . . .   | 69  |
| Data Set Naming Conventions . . . . .                     | 26 | Major Time Slices . . . . .   | 70  |
| Data Entry . . . . .                                      | 27 | Minor Time Slices . . . . .   | 72  |
| Creating Data Sets . . . . .                              | 27 | <b>System Implementation</b> . . . . .  | 75  |
| Entry Modes for EDIT . . . . .                            | 28 | Tailoring a Message Control Program . . . . .                                       | 75  |
| Input Mode . . . . .                                      | 28 | Mixed Environment MCPs . . . . .  | 75  |
| Edit Mode . . . . .                                       | 28 | TSO-Only MCP . . . . .  | 76  |
| Modifying Data Sets . . . . .                             | 28 | LINEGRP Macro Instruction . . . . .   | 77  |
| Data Set Management Commands . . . . .                    | 29 | LISTTA Macro . . . . .  | 81  |
| TSO Data Utilities . . . . .                              | 29 | TSOMCP Macro . . . . .  | 82  |
| Text-Handling . . . . .                                   | 29 | Writing Cataloged Procedures for TSO . . . . .                                      | 86  |
| Data Set Manipulation . . . . .                           | 30 | Message Control Program . . . . .   | 86  |
| Compiling and Executing Programs . . . . .                | 30 | Time Sharing Control Task . . . . .   | 87  |
| Remote Job Entry . . . . .                                | 32 | Background Reader (BRDR) . . . . .  | 89  |
| System Control . . . . .                                  | 32 | TSO Trace Writer . . . . .  | 90  |
| User Authorization . . . . .                              | 32 | Logon Cataloged Procedure . . . . .   | 91  |
| System Operation . . . . .                                | 32 | TSO System Parameters . . . . .   | 92  |
| Command Procedures . . . . .                              | 33 | The Time Sharing Control Task Parameters . . . . .                                  | 92  |
| Other Commands . . . . .                                  | 34 | Driver Parameters . . . . .   | 93  |
| <b>Programming at The Terminal</b> . . . . .              | 35 | Buffer Control Parameters . . . . .   | 95  |
| COBOL . . . . .   | 36 | System Parameter Format . . . . .   | 96  |
| Entering the Source Program . . . . .                     | 36 | Tuning a System With TSO . . . . .  | 102 |
| Compiling a COBOL Program . . . . .                       | 37 | Using TSO Trace . . . . .   | 106 |
| Program Execution . . . . .                               | 38 | Writing Installation Exits for the SUBMIT Command . . . . .                         | 108 |
| Interactive Programs . . . . .                            | 38 | Writing Installation Exits for the OUTPUT, STATUS,<br>and CANCEL Commands . . . . . | 110 |
| A COBOL Example . . . . .                                 | 39 | Writing a LOGON Pre-Prompt Exit . . . . .   | 111 |
| FORTRAN . . . . .   | 41 |   |     |
| Entering the Source Program . . . . .                     | 41 |   |     |

|   |     |  |     |
|---|-----|--|-----|
| <b>Storage Estimates</b> . . . . .                | 115 | Remote Job Entry . . . . .                                 | 121 |
| Main Storage Requirements . . . . .               | 115 | System Control . . . . .                                   | 121 |
| MVT Basic Fixed Requirement . . . . .             | 115 | Session Control . . . . .                                  | 121 |
| Nucleus . . . . .                                 | 115 |  |     |
| Master Scheduler Region . . . . .                 | 115 | <b>Appendix B: Program Products</b> . . . . .              | 122 |
| Link Pack Area . . . . .                          | 116 |  |     |
| System Queue Area . . . . .                       | 116 | <b>Appendix C: Driver Entry Codes</b> . . . . .            | 124 |
| Message Control Program Requirement . . . . .     | 116 |  |     |
| Time Sharing Control Region Requirement . . . . . | 116 | <b>Appendix D: Terminal Message Requiring Installation</b> |     |
| Dynamic Area Requirements . . . . .               | 117 | <b>Action</b> . . . . .                                    | 128 |
| Foreground Region Requirement . . . . .           | 117 |  |     |
| Auxiliary Storage Requirements . . . . .          | 117 | <b>Appendix E: Glossary</b> . . . . .                      | 140 |
| Swap Data Sets . . . . .                          | 117 |  |     |
| System Libraries and Data Sets . . . . .          | 118 | <b>Bibliography</b> . . . . .                              | 148 |
|   |     | <b>Index</b> . . . . .                                     | 149 |
| <b>Appendix A: TSO Commands</b> . . . . .         | 119 |  |     |
| Data Management . . . . .                         | 119 |  |     |
| Language Processors . . . . .                     | 120 |  |     |
| Program Control . . . . .                         | 120 |  |     |

## Figures

|   |    |   |     |
|---|----|---|-----|
| Figure 1. Simple Identification Scheme . . . . .                            | 25 | Figure 27. Terminal Monitor Program . . . . .   | 61  |
| Figure 2. User Identification Hierarchy . . . . .                           | 25 | Figure 28. Service and TEST Routine . . . . .   | 63  |
| Figure 3. Example of Data Set Naming Conventions . . . . .                  | 27 | Figure 29. TCAM Message Control Program . . . . .                                     | 66  |
| Figure 4. Program Control Commands . . . . .                                | 31 | Figure 30. System Overview . . . . .  | 67  |
| Figure 5. A Command Procedure . . . . .                                     | 33 | Figure 31. Typical Main Storage Map . . . . .   | 68  |
| Figure 6. Entering a COBOL Program . . . . .                                | 36 | Figure 32. Queue Service Time . . . . .   | 69  |
| Figure 7. Compiling a COBOL program . . . . .                               | 37 | Figure 33. Minor Time Slice . . . . .   | 70  |
| Figure 8. Defining the Terminal as a File . . . . .                         | 38 | Figure 34. Job Stream to Tailor MCP . . . . .   | 77  |
| Figure 9. Terminal Session Creating a COBOL<br>Program . . . . .            | 40 | Figure 35. Sample MCP . . . . .   | 85  |
| Figure 10. FORTRAN Syntax Checker Diagnostic . . . . .                      | 42 | Figure 36. Sample MCP . . . . .   | 85  |
| Figure 11. Sample of FORTRAN Compiler Output . . . . .                      | 42 | Figure 37. Sample MCP Start Procedures . . . . .                                      | 87  |
| Figure 12. A Command Procedure to Invoke the<br>PL/I (F) Compiler . . . . . | 46 | Figure 38. Sample Cataloged Procedure to Start<br>Time Sharing Control Task . . . . . | 88  |
| Figure 13. Use of a Command Procedure . . . . .                             | 47 | Figure 39. Sample Background Reader (BRDR)<br>Procedure . . . . .                     | 90  |
| Figure 14. Implicit use of Procedure . . . . .                              | 47 | Figure 40. Sample TSO Trace Start Procedure . . . . .                                 | 90  |
| Figure 15. A Command Procedure to Invoke a User<br>Program . . . . .        | 47 | Figure 41. Sample LOGON Cataloged Procedure . . . . .                                 | 92  |
| Figure 16. A Command Procedure for a<br>Compile-Load-Go Sequence . . . . .  | 48 | Figure 42. TSO System Parameter Syntax (Part 1 of 4) . . . . .                        | 97  |
| Figure 17. Using a Compile-Load-Go Command<br>Procedure . . . . .           | 48 | Figure 43. Sample TSO System Parameters . . . . .                                     | 101 |
| Figure 18. ITF: BASIC Sample Session . . . . .                              | 50 | Figure 44. Summary of Tuning Objectives for Various<br>Techniques . . . . .           | 102 |
| Figure 19. ITF: PL/I Sample Session . . . . .                               | 52 | Figure 45. Sample Use of the UNITNAME Macro<br>Instruction . . . . .                  | 103 |
| Figure 20. Code and Go FORTRAN Sample Session . . . . .                     | 54 | Figure 46. SVC Modules useful for Residence in TSO . . . . .                          | 105 |
| Figure 21. TSO Control Flow Diagram . . . . .                               | 55 | Figure 47. Sample Job System to Run TSO Trace<br>Data Set Processor . . . . .         | 106 |
| Figure 22. The Time Sharing Driver . . . . .                                | 57 | Figure 48. Format of the TS Trace Data Set . . . . .                                  | 107 |
| Figure 23. The Time Sharing Control Task . . . . .                          | 58 | Figure 49. Portion of Sample PL/I Logon Pre-Prompt<br>Exit . . . . .                  | 114 |
| Figure 24. The Region Control Task . . . . .                                | 59 | Figure 50. Swap Allocation Unit Sizes . . . . .                                       | 117 |
| Figure 25. The LOGON/LOGOFF Scheduler . . . . .                             | 60 |   |     |
| Figure 26. LOGON Linkage . . . . .  | 61 |   |     |

**Summary of Amendments  
for GC28-6698-4  
as Updated by GN28-2519  
Component Release 360S-OS-586**

**Dynamic Specification of DCB Parameters**

The ATTRIB command, which provides the TSO user with the capability for assigning data set attributes (DCB parameters) dynamically from the terminal, has been added.

**Summary of Amendments  
for GC28-6698-4  
OS Release 21**

**Publications Change**

Information describing job priorities, dispatching, and classes, formerly in **IBM System/360 Operating System: Concepts and Facilities**, GC28-6535, is now contained in **IBM System/360 Operating System: Introduction**, GC28-6534.

**SVC Restrictions**

**Programming Change**

SVC 93 (TGET/TPUT) can be used for certain background purposes.

**Dynamic Allocation**

**Programming Change**

A dynamically allocated data set uses 50 tracks for a primary allocation and 10 for secondary allocation.

**Time Sharing Driver Parameters**

**Rewritten section**

The description of the Time Sharing Driver start parameters has been rewritten to show dependencies between parameters.

**Tuning the TSO system**

**New Section**

A new section has been added to describe TSO tuning techniques.

**Bibliography**

**New Section**

A discussion of related reading, formerly in the Preface, has been moved to the Bibliography and rewritten.

**Swap Data Sets**

**Clarification**

Swap data sets must be allocated along cylinder boundaries.

**Summary of Amendments  
for GC28-6698-3  
as Updated by GN28-2502  
OS Release 20.6**

TSC START PARAMETERS

Programming Change

Two new parameters, TSCREGSZ and DUMP have been added. TSCREGSZ allows an installation to specify a size for the TSC region. DUMP reserves the swap data sets so they are not initialized during a restart.

START COMMAND

Programming Change

Most of the TSC start parameters can be altered by the operator using the START command.

MISCELLANEOUS CHANGES

Swap Data Sets

Swap data sets must begin on a cylinder boundary. Parallel data sets must reside on the same device types.

**Summary of Amendments  
for GC28-6698-3  
as Updated by GN28-2497  
OS Release 20.1**

Teletype ASR

Use of paper tape reader/punches attached to Teletype ASR terminals are not supported.

Restrictions to Foreground Programs

Tape and multivolume data sets are not supported by most Command Processors and cannot be dynamically allocated.

TSOMH Macro Instruction

The NOLOG operand has been removed.

2260, 2265

The SCRFSIZE operand has changed to SCREEN. The valid screen sizes are 12x80, 12x40, 6x40, and 15x64.

DIAL Operand

DIAL=YES is default for TERM=3335.

FEATURE Operand

Two defaults have changed.

Background Reader for SUBMIT Command

The output of the background reader for the SUBMIT command is placed on SYS1.SYSJOBQE.

ADDR Operand

Only one station identification character can be specified for each non-switched line with a 1050 terminal attached to it.

OS PL/I Checkout Compiler

The titles of publications describing the checkout compiler are added.

Swap Data Sets

IBM 2311 is not supported as a swap device.

Broadcast Data Set

The broadcast data set contains a list of valid users and should not be password protected.

EDIT Default Line Pointer

The default line pointer is positioned at the last line referenced.

FORTRAN

The EDIT FORT operand has no default.

Relative Line Number

The order of relative line numbers is determined by the MCP start cataloged procedure.

Defining a New UADS

The new UADS should be defined with a file name of SYSUADS and a data set name other than SYS1.UADS.

Storage Estimates

The MCP storage requirements are increased.

Drive Entry Codes CLASS

CLASS refers to the CODES parameter of the TSO Trace Data Set Processor.

**Summary of Amendments  
for GC28-6698-3  
OS Release 20.1 (June)**

TMP Step Library

A discussion of the advantages of concatenating Command Library to SYS1.LINKLIB, the Linkage Library.

OS PL/I Checkout Compiler

A discussion of the uses and facilities of the Checkout Compiler.

2260, 2265

The macro instructions for generating the MCP have additional operands for 2260 and 2265 support.

**Summary of Amendments  
for GC28-6698-2  
OS Release 20.1 (March)**

System Implementation

A section has been added describing the techniques used in implementing a TSO system. This section consists of discussions of:  
Generating a Message Control Program  
Writing Cataloged Procedures Used with TSO  
Specifying TSO System Parameters  
Tuning the Time Sharing Driver  
Writing Installation Exits for:  
The SUBMIT Command  
The OUTPUT, STATUS, and CANCEL Commands  
The LOGON Command

Storage Estimates

Most of the information in the Storage Estimates section, including the sample TSO configuration, has been deleted and moved to the publication IBM System/360 Operating System: Storage Estimates, GC28-6551.

Driver Entry Codes

An appendix has been added containing the format and meaning of the TSEVENT macro instructions used to notify the Time Sharing Driver of system events.

Terminal Messages Requiring Installation Action

An appendix has been added containing messages which when received at a terminal requires the installation to take certain actions.

Message Control Program Assembly Error Messages

An appendix has been added containing the text of error messages generated by the macro instructions used to generate the Message Control Program.

## Introduction

The IBM System/360 Operating System Time Sharing Option (TSO) adds general purpose time sharing to the facilities already available through the MVT configuration of the control program. As a result, the system provides a number of new capabilities:

- It gives users access to the system through a **command language** which is entered at **remote terminals** -- typewriter-like keyboard-printer or keyboard-screen devices connected through telephone or other communication lines to the computer.
- It gives those who may not be programmers the use of data entry, editing, and retrieval facilities.
- It makes the facilities of the operating system available to programmers at remote terminals to develop, test, and execute programs conveniently, without the job turnaround delays typical of batch processing. Both terminal-oriented and batch programs can be developed at terminals.
- It allows the management of an installation to dynamically control the use of the system's resources from a terminal.
- It creates a time-sharing environment for terminal-oriented applications. Some applications, such as problem-solving languages, terminal-oriented compilers, and text-editing facilities, are available as IBM Program Products. Installations can add others suited to their particular needs.

A major consideration in the design of TSO is ease of use. The way in which a user communicates with the system can be kept simple to encourage people who may not be programmers to take advantage of the speed and versatility of a computing system to solve their problems. There are four ways in which TSO achieves this goal:

- The physical medium is easy to use. Most users are already familiar with the conventional typewriter keyboard. Information is easy to enter through the terminal's typewriter-like keyboard, and no complex procedures are required to obtain output from the computer.
- The way in which a terminal user defines his work is uncomplicated. He enters commands which resemble English language words to describe the general function he wants to accomplish. If the user chooses, he can create his own commands and command system.
- If a user doesn't know how to define his work to the system, he can type **HELP** and receive information pertinent to the type of operation he is trying to perform. In cases, he doesn't need to enter detailed parameters describing every aspect of the work he is doing; the system uses default values that are appropriate for most jobs. If he fails to provide parameters the system needs to do the work he requested, the system will ask him for the missing information, item by item, by "prompting" him for it in a conversational way.
- The system keeps the terminal user aware of what is happening, so he knows what to do next. He "converses" with the system on a step-by-step basis. The system lets him know when it is ready to accept input from him, and it tells him immediately when there has been change in the status of his program. If the user receives a message he doesn't understand, he can request more information about the situation simply by typing a question mark. The messages he receives use uncomplicated language to describe the situation. When the messages become familiar to him, he may request the system to use the abbreviated messages that are available with some of the programming languages.

## Advantages of a Time Sharing System

In a simple batch processing system, one job at a time has access to the resources of the system (main storage, the central processing unit, and I/O equipment). A programmer's job is loaded into the computer and its operation is controlled by the system operator. The job acquires the resources it needs as it runs to completion; resources the job doesn't need are unused. When the job is finished, results are produced, a new job is loaded and executed, and the output for the completed job (for example, a printout) is sent to the programmer. An inherent problem with this type of processing is **turnaround time**, the elapsed time between the submission of a job to the computer center for processing and the return of results to the programmer. Another problem is the inefficient use of resources.

In a multiprogramming system (e.g., a system that operates under the control of the MVT configuration of the System/360 Operating System), several jobs share the resources of the system concurrently, so the use of resources is much more efficient. Although jobs are processed faster, the operator at the system console still controls the system, and the programmer still must wait for results to be returned to him.

A time sharing system reduces delays in receiving results. A larger number of jobs share the resources of the system concurrently, and the execution of each job is controlled primarily by a remote terminal user. Thus, **time sharing** can be defined as the shared, conversational, and concurrent use of a computing system by a number of users at remote terminals.

The system resources shared by the time sharing jobs (foreground jobs) entered from the terminals are also shared by batch jobs (background jobs) that are being processed at the same time. Each foreground main storage region handles many active foreground jobs, although only one job is actually in the region at any moment in time. A foreground job is assigned to a main storage region and has access to the system's resources for a short period of time called a **time slice**. The other foreground jobs assigned to that region are saved on auxiliary storage while the job being executed in main storage receives a time slice.

At the end of the job's time slice, or if the job enters the wait state for terminal I/O, the main storage image of the job (that is, programs, work areas, and associated control blocks) is stored on a direct access device and another job is brought into the same region of main storage and given a time slice. TSO schedules a similar time slice for each ready foreground job. The apportionment and duration of time slices is discussed in detail in the "System Summary" section of this manual.

The process of copying job images back and forth between main and auxiliary storage is called **swapping**. Writing an image to auxiliary storage is a **swap out**; reading one into main storage is a **swap in**.

All foreground jobs are assigned the same priority. The order in which foreground and background jobs are processed is determined by the operating system task dispatcher and the TSO control routines. Job priorities, job classes, and the dispatching of tasks are discussed in **IBM System/360 Operating System: Introduction, GC28-6534**.

The apportionment of slices of processing time to foreground jobs is not apparent to a terminal user. At the terminal, the response of the system to requests for action is fast enough so that he has the impression that he is the sole user. As far as the user is concerned the distinctive feature of a time-sharing system is the way in which it "converses" or interacts on a step-by-step basis with him as he does his work. He is prompted for information the system needs to execute his job, he receives immediate response to his requests for action, and he is notified immediately of errors the system detects, so that he can take corrective action at once.

In general then, a time-sharing system differs from a batch processing system in three ways:

1. A terminal user concurrently shares the resources of a computing system with other terminal users.
2. A terminal user can enter his problem statements and other input into the system as he develops them, and he receives immediate results. Thus the problem of turnaround time (the amount of time between when he submits his job for processing and when he receives results) inherent with batch job operations is greatly reduced.
3. A terminal user is constantly aware of the progress of his job. He requests results from the system one step at a time, he is prompted for any additional information the system requires, he receives immediate notification of the status of his work, and he is apprised of errors as soon as the system detects them. The terminal user can change his problem statements or correct errors immediately after entering each statement or at any time during the current terminal session. Thus, he minimizes the need for reruns.

## Using a Terminal

A terminal session is designed to be an uncomplicated process for a terminal user: he identifies himself to the system and then issues commands to request work from the system. As the session progresses, the user has a variety of aids available at the terminal which he can use if he encounters any difficulties.

Commands specifically tailored to an installation's needs can be written and added to the command language or used to replace IBM-supplied commands.

### Starting and Stopping a Terminal Session

When the user has some work to perform with the system, he dials the system number if he has a terminal on a **switched line**, or he turns the power on if he has a terminal on a **non-switched line**. A switched line is one in which the connection between the computer and a terminal is established by dialing the system's number from the terminal. A non-switched line is one with a connection between the computer and a terminal. With an IBM 2741 terminal or an IBM 1050 terminal, the system responds by unlocking the keyboard. In any case, the user identifies himself by entering "LOGON" and one or more of the following fields:

- A **user identification**, for example, the user's name or initials, which the system will use to identify his programs and data sets.
- A **password**, assigned by his installation, usually known only to the user and the system manager.
- An **account number**, which defines the account in which his system usage totals are to be accumulated.
- A **LOGON procedure name**, which identifies a cataloged procedure that specifies what system resources he will be using.

The user may omit the last three fields if the system manager has defined only one account number and LOGON procedure for him and no password is used.

The LOGON processor verifies that the user is an authorized TSO user, then checks the password, if it is required, and account number in a record it keeps of user attributes, called the User Attributes Data Set (UADS). From the attributes, the LOGON command operands, and a LOGON cataloged procedure, the system builds a user profile, which is used to control the processing of his job. The system assigns the user's job to a time-sharing (foreground) region of main storage and allocates other resources, such as auxiliary storage space and user data sets according to the LOGON procedure.

LOGON marks the start of a terminal session. When the user completes his work, he enters "LOGOFF" to end the session. The system then updates his job's system use totals, releases resources allocated to it, and releases the terminal from TSO. A session is also terminated any time the terminal user enters LOGON to start a new session. In this case, the old session is terminated and a new one is begun; the terminal is not released in the process.

### **Working at the Terminal**

The user enters commands to define and execute his work at the terminal. He enters a command by typing a command name, such as EDIT and possibly some additional operands. The system finds the appropriate operands. The system finds the appropriate command processor--a load module in a command library--and brings it into the foreground region assigned to the user for execution. For example, in response to entering the EDIT command, the system brings in the EDIT command processor, the data handling routine used to create and update data sets.

If a user does not enter all the operands associated with a particular command name, default values are assumed where possible. If necessary operands are missing, the system prompts the user for them with a message such as "ENTER DATA SET NAME." The user can reply with the missing value, or enter a question mark for a further explanation of what the system needs. If the user chooses, he can specify that prompting messages be suppressed.

A terminal user can also receive assistance through the HELP facility. He can request information regarding the syntax, operands, or function of any command, subcommand, or operand. If he enters HELP followed by a command name, he receives an explanation of the command and the operands required with it. HELP followed by a subcommand name furnishes an explanation of the subcommand if the user is working with the command at that time. Entering HELP by itself returns a description of the command language, a list of the commands, and an explanation of how to use HELP to obtain further information.

During a typical session, the user enters a series of commands to define and perform his work. If the sequence is one that is used often, he can store the sequence in a data set and then execute the sequence whenever he needs it by entering the EXEC commands.

The commands provided with the system handle data and program entry, program invocation in either the foreground or the background, program testing, data management, and session and system control. IBM Program Products are available to support problem solving, data manipulation, and text formatting, to provide terminal-oriented language processors, and to make these processors more convenient to use from the terminal.

## System Configuration

TSO is an extension of the MVT configuration of the control program on System/360 Models 50 through 195, or System/370 Models 145, 155, and 165. TSO also operates with the Model 65 Multiprocessing (M65MP) configuration. The minimum machine configuration for System/360 models must include 384K of main storage, the required I/O devices for MVT, plus at least one each of the following:

- A terminal (IBM 1050, 2741, 2260 Local or Remote, 2265, or Teletype<sup>1</sup> Model 33 or 35 KSR and ASR).
- A transmission control unit (IBM 2701, 2702, or 2703), unless all terminals are locally attached 2260 Display Stations.
- Sufficient direct access storage space (IBM 2301, 2311, 2303, 2305, 2314, or 3330) for command libraries and system data sets.
- Sufficient direct access storage space for swap data sets.

In a System/360 with 384K of main storage, TSO is, in effect, a "dedicated" time sharing system. In other words, with 384K the system can run as a time sharing system or as a batch job processing system, but not both at the same time. To run both time sharing and batch jobs concurrently or to execute on M65MP or System/370 models, at least 512K of main storage is required. At least 128K of main storage is required for system generation.

### Terminals

Some remote terminal suitable for interactive applications have keyboards for entering input data and either typewriter-like printers or display screens. A remote terminal incorporates or is attached to a control unit. The control unit is in turn connected to the system by either of two ways:

- Through a device such as a data set to a dialed (switched) line to the system.
- Through either a direct or a leased line to the system.

At the computer site the communication line connects to a Transmission Control Unit, which in turn is attached to one of the computer system's multiplexor channels. The IBM 2260 Display Station can be an exception to this general configuration. Its control unit, the IBM 2848 Display Control, can be attached directly to a multiplexor or selector channel. This mode of operation is called local attachment.

TSO uses the Telecommunications Access Method (TCAM) for terminal access. TSO provides terminal handling programs for the following terminals:

- IBM 2741 Communication Terminal.
- IBM 1050 Printer-Keyboard.
- Teletype<sup>1</sup> Model 33 and 35 KSR or ASR. (Paper tape is not supported with Teletype<sup>1</sup>.)
- IBM 2260 and 2265 Display Stations.

---

<sup>1</sup>Trademark of Teletype Corporation, Skokie, Illinois.

The IBM 2741 Receive Interruption Feature and the Transmit Interruption Feature are recommended for use with the 2741. These features are described in the publication **IBM 2741 Communications Terminal**. The Transmit Interrupt, Receive Interrupt, and Text-Timeout Suppression features are recommended for use with the IBM 1050. 1050 multidrop is not supported. These features are described in the publication **IBM 1050 System Summary**. Note that some of these features are not available through the IBM 2701 Data Adapter Unit.<sup>2</sup>

### **Transmission Control Unit**

TSO requires at least one of the following transmission control units to handle terminal communications:

- IBM 2701 Data Adapter Unit.
- IBM 2702 Transmission Control.
- IBM 2703 Transmission Control.

The Terminal Interruption Features are recommended for use with the 2702 and 2703 transmission control units and must be present if the terminals are to use the features. These units are described in the following publications.

- **IBM 2701 Data Adapter Unit, Component Description, GA22-6864.**
- **IBM System/360 Component Description, IBM 2702 Transmission Control, GA22-6846.**
- **IBM 2703 Transmission Control, Component Description, GA27-2703.**

### **Swap Data Set Devices**

The process of copying images back and forth between main and auxiliary storage is called **swapping**. Writing an image to auxiliary storage is a **swap out**; reading one into main storage is a **swap in**. The pre-formatted data sets into which jobs are written are called **swap data sets**. A swap data set is divided into **swap allocation units**, each of which consists of a device-dependent number of 2048-byte records. An integral number of swap allocation units, not necessarily contiguous, are assigned to each job to contain the swapped out image of the job.

If there is more than one foreground region, they share the available swap data sets, but the cycle of swapping for each region is essentially independent of other regions. However, the system avoids queueing on swap data sets if possible.

TSO requires sufficient storage capacity on one or more of the following for swap data sets:

- IBM 2301 Drum Storage.
- IBM 2303 Drum Storage.
- IBM 2305 Fixed Head Storage, Model 1 or 2.
- IBM 2314 Direct Access Storage Facility.
- IBM 3330 Disk Storage Facility.

See the Storage Estimates section of this publication for information on swap data set sizes.

The record overflow feature is required for the devices used to store the swap data sets. One or more data sets on any of the above devices can be used for swap data sets.

---

<sup>2</sup>Terminals which are equivalent to those explicitly supported may also function satisfactorily. The customer is responsible for establishing equivalency. IBM assumes no responsibility for the impact that any changes to the IBM-supplied products or programs may have on such terminals.

The direct access storage space required for the swapped data may be divided among the devices listed above in either of two ways. The user may specify that swapped data be distributed serially among a hierarchy of data sets, or he may specify parallel distribution of data onto two or more devices. With serial distribution, the first data set in the hierarchy is filled with swapped data, and then the next data set in the hierarchy is used. For example, a drum, because of its higher access speed, could be assigned as the first unit in the hierarchy, with a 2314 assigned to receive any overflow of swapped data.

With the parallel distribution scheme, two or more devices are used concurrently to receive swap data sets. Identical device types must be used with parallel swapping. The exact order in which data sets are written on either of the devices is determined by the system, based on the I/O activity taking place in the channel at the time of a swap out. For example, if the two data sets are on devices on separate channels, swap performance improves substantially.

Before a terminal job can be swapped out of main storage, activity associated with the job must be brought to an orderly halt. The halt must be performed in such a way that the job is not aware of it, and information must be saved to restart the job when its next time slice is scheduled. The orderly suspension of a job's activity before a swap out is called **quiescing the job**. Quiescing includes the removal of the majority of the control blocks associated with the job from the system queues so they can be written to the swap data set along with the contents of the main storage region assigned to the job.

## **The Relationship of TSO to the Operating System**

For the data processing center, TSO is compatible with operating system standard formats and services, while providing the flexibility needed for various time sharing and terminal-based applications.

TSO is not necessarily intended to be used as a dedicated time-sharing system, that is, a system on which only time-sharing operations take place. TSO augments the facilities already available with the operating system: batch processing, teleprocessing, and other data processing activities can take place concurrently on the same system.

Once an installation has generated a system that includes TSO, time sharing operations can be started and stopped at any time by the system console operator. The operator can specify how many regions of main storage are to be assigned to time sharing users. Each region can serve many users, whose programs are swapped back and forth between main and auxiliary storage. Time sharing, or foreground operations, can take place concurrently with batch or background operations. (Background jobs are not swapped.) If the user chooses, he can dedicate his system to time sharing and run only foreground jobs. If there are periods when TSO is not needed in the system, time sharing operations can be stopped, and the system will then process background jobs in the usual way with MVT and TCAM.

Terminal communications are handled by the Telecommunications Access Method (TCAM) through an interface that allows the use of standard sequential access method I/O statements and macro instructions.

All of the MVT facilities are available to a background job. Foreground jobs can use most of the operating system access methods for data set access (e.g., BSAM, QSAM, BDAM etc.). All devices available to these access methods are usable by foreground jobs. A detailed list of restrictions is in the "Restrictions and Limitations" section of this manual.

## **Execution of Background Jobs from the Terminal**

In addition to the foreground execution of programs, TSO allows jobs to be submitted for execution in the background, or batch, portion of the system. If his installation authorizes it, a user can submit a background job at his terminal, be notified of the job's status, and then receive results of the job at the terminal. If he chooses, he can specify that the output of his job be produced at the computing center, rather than at the terminal.

## **Foreground-Background Compatibility**

Because time sharing is carried out within the framework of MVT job and task management, the foreground and background environments are compatible. TSO uses the same data formats, programming conventions, and access methods as the rest of the operating system. The programming languages and service programs available with TSO are compatible with their background counterparts.

The TSO command language is also generally compatible with the Conversational Remote Job Entry (CRJE) command language. Programs can be developed in the foreground and stored in background libraries. These programs are compatible with other system programs. Most problem programs can be executed in either the background or the foreground without revision or recompilation.

## **Restrictions and Limitations**

Certain facilities are unavailable to foreground jobs, although they remain available to background jobs. These include:

- The BTAM and QTAM telecommunications access methods.
- The Graphics Access Method (GAM).
- The EXCP equivalents of the BTAM, QTAM, and GAM access methods.
- Main storage requests for hierarchy 1 (all foreground requests for main storage are allocated to hierarchy 0).
- Use of Job Control Language in the foreground for other than single-step jobs (the TSO command language is used to provide the equivalent of multi-step jobs).
- Checkpoint/Restart Facility (foreground requests for checkpoint are ignored).
- Rollout/Rollin Option.
- TESTRAN Facility.
- Multivolume and tape data sets are not supported by most Command Processors and cannot be allocated dynamically.

SVC numbers 92 through 102 (decimal) are added to the system for TSO. The following SVCs can be issued by problem programs in the foreground region:

- SVC 93--TGET/TPUT (execute terminal I/O).
- SVC 94--STCC (specify terminal control characteristics).
- SVC 95--TSEVENT (notify the supervisor of an event).
- SVC 96--STAX (specify a terminal attention exit).

- SVC 97--Breakpoint (used by TEST command).
- SVC 98--PROTECT (protect a data set with a password).
- SVC 99--Dynamic Allocation (of a data set).
- SVC 100--Submit a job to the background.
- SVC 102--AQCTL -- used by TCAM to communicate with problem programs.

Of these, only SVC 98--PROTECT--can be issued by programs executing in the background. SVCs 92 (TCB EXCP) and 101 (TCAM-TSO Communication) are used only by supervisor programs.

Including TSO in a system adds no restrictions to programs executed in the background. For example, other teleprocessing applications can be run simultaneously.

### System Control

The management of an installation can shift most of the responsibility for controlling the time sharing system from the operator at the system console to users at remote terminals, called **control terminals**. A control terminal user can alter the system configuration to meet changing work loads. For instance, he can assign an extra region during peak periods, and then release it to be used for batch operations during slack periods. Such changes require no shutdown of TSO and are not noticed by the users of other regions. Even the starting and stopping of TSO operations is accomplished without shutting down the system or affecting background operations.

### Job Definition and Scheduling

To the operating system, each terminal session from LOGON to LOGOFF is one terminal job, corresponding to a single step batch job. The job control statements that define a terminal job are stored in the LOGON procedure used to begin the session. The "EXEC" JCL statement in the LOGON procedure identifies the program the user wants loaded into his region for execution. The program may be the TSO-provided command language handler or an installation provided application program.

An important feature of TSO is the dynamic allocation of data sets for time sharing users. A user may defer definition of his data sets until he requires them. During LOGON processing, any data sets named on Data Definition (DD) statements in the procedure are allocated to the terminal job. Any data sets requiring volume mounting by the operator, must be defined here. The procedure also includes dynamic DD statements (similar to a DD DUMMY), which reserve control block space for data sets the user may allocate during the session. The dynamic allocation facility allows data sets to be created, deleted, concatenated, or separated without previous allocation at the beginning of the job step.

### Tuning the Time Sharing System

In a time sharing system, execution time is divided among the active foreground jobs and background jobs in brief time slices. A time slice must be long enough to perform a meaningful amount of processing, but not so long that the time between successive slices prevents quick response to conversational users. At the same time, time slices cannot be so short and frequent that system overhead for swapping and task switching becomes excessive. Balancing these factors depends on the number and type of jobs the system is processing. A solution for one job mix is not necessarily suitable for another job mix. The **TSO time sharing algorithms** -- the formulas used to calculate the division of time among jobs -- are based on several variables, most of which can be specified by the installation to tune the system for their particular workload. Some of the tuning

variables such as the number of foreground regions and the maximum number of users, can be set or modified by the system operator or a user at a control terminal whenever the system is running. Others are specified as parameters in SYS1.PARMLIB. These parameters are used when the operator starts the time sharing operations.

The time sharing algorithms are described in detail in the "System Summary" section of this manual. They are implemented by a subroutine called **Time Sharing Driver**. The Driver makes decisions about system functions such as swapping and task switching. An installation may experiment with other time sharing algorithms by modifying or replacing the driver, and specifying use of the new Driver in the SYS1.PARMLIB parameters used when the operator starts time sharing operations.

Execution time may also be affected by the choice of modules to be included in the Link Pack Area (LPA) extension in the Time Sharing Control Task (TSC) region. The size of the LPA extension and the amount of main storage dynamically allocated by the driver are major factors in determining the size of the TSC region. The installation may let the TSC calculate its own region size or may specify a TSC region size, either in SYS1.PARMLIB or on the START command used to start TSO, to compensate for additional main storage requirements created during tuning.

## Monitoring System Use and Performance

By extending the services of the system to many concurrent users, TSO makes the operating system more useful to more people. However, installation management's job of monitoring system use and performance becomes more complex. Three tools are provided to help management maintain a clear picture of what the system is doing.

**System Management Facilities (SMF):** The SMF Option can be used with TSO. Both the data collection and dynamic control facilities are extended to the foreground environment.

With the data collection facility, records describing both the system environment and individual user activity are written to the SMF data sets in a format similar to that used for background records. The system environment data includes:

- Machine configuration.
- Resource status.
- Library management information.

This information is recorded whenever time-sharing operations are started, modified, or stopped by an operator. The user data includes:

- I/O device use.
- Data set use.
- Main storage used.
- Time resident in main storage.
- Time actually spent executing.

The user data is recorded at LOGON and LOGOFF and during a terminal session whenever a user changes the status of his data sets with the dynamic allocation facility. The information on the use of data sets is particularly useful to the installation for controlling the use of secondary storage in the time-sharing environment.

The SMF dynamic control exits give the installation access to control program information at key points during the processing of jobs, including foreground jobs. The step initiation and termination exits are taken, if present, when a user begins or ends a terminal session. These routines can record information and control processing for foreground jobs just as they do for background jobs. SMF is discussed in detail in the publication **IBM System/360 Operating System: System Management Facilities**, GC28-6712.

An additional installation exit, separate from the SMF dynamic control exits, is provided from the routine handling user LOGON. This exit allows the installation to establish its own user verification and control procedures, independent of those supplied with the system. The section of this publication 'Writing a Logon Pre-prompt Installation Exit' describes the parameters passed and what actions the exit may take.

**MONITOR Command:** The MONITOR command allows the operator to watch the changing workload on the system over a period of time. In addition to the job initiation, data set, and volume information formerly available with the DISPLAY command, he can request notification of time-sharing users logging on and off the system. The DISPLAY command now gives the system workload at a particular point in time, and has been extended to include information relative to the time-sharing environment, such as the number of foreground regions and the number of active terminals. Both MONITOR and DISPLAY, like other operator commands concerned with the time-sharing operation, are available to a control user at a remote terminal as well as the system operator at the console.

**TSO Trace Program:** The TSO Trace Writer Program provides a detailed history of what the system does over a period of time. The Trace Program records a stream of information that all components of the system are continuously passing to the Time Sharing Driver. The Driver uses this information in its calculations of resource allocation. When the operator starts the Trace Program, it intercepts these event signals and records them with a time stamp in a data set. Typical events recorded are "job requesting terminal input" and "swap completed." The TSO Trace Data Set Processor can be used at a later time to format and print out the information recorded by the Trace Program. The Trace Data Set Processor can be requested to list only those events associated with a particular component of the system, such as the dispatcher, or to list only those events associated with a particular terminal or set of terminals. Using this information, system management can determine how well the system is responding to the workload and make adjustments to the tuning variables if necessary.

## **System Security**

The need for adequate data and program protection is increased in the time-sharing environment, where many persons are simultaneously using the system. The system itself must be protected against unauthorized users. Each user's programs and data must be protected against accidental destruction by other users. Confidential data must be safeguarded against unauthorized access.

### **User Verification**

Any user starting a terminal session is required to supply a user identification recognized by the system; that is, one that has been defined by the system administrator. The installation may also require the user to supply a unique and confidential password with the LOGON command.

Further verification of a user's identity can be performed by the optional installation routine called when a user logs on. This routine can request further information from the applicant and deny him access to the system if he fails to provide it.

## **Program Protection**

Although a number of users may have jobs assigned to the same foreground main storage region, only one user's job is present in the region at a particular time -- the other jobs are temporarily stored in the swap data sets. No user can accidentally destroy or tamper with another user's job. Like the background regions under MVT, the foreground regions have unique storage keys, preventing a job from modifying any area of main storage outside its assigned region.

## **Data Set Security**

Because any user can refer to any data set in the system catalog, the data set security facility of the operating system is extended to allow individual users to protect their own data sets from unauthorized reference. A user can assign one or more passwords to a data set. If anyone subsequently attempts to open the data set, he is prompted for the password(s). If he fails to supply the correct password in two attempts, his program is terminated.

The password assigned to a data set can be the one associated with the user for LOGON. In this case, the user will not be prompted for the password when opening his own data set. Any other user, however, must supply the correct password to refer to that data set.

Passwords can be assigned for two levels of protection:

- **Modification protection.** No password is required to open the data set for reading, but a password must be supplied to write into the data set, or to delete it. This type of protection is required for system libraries and data sets, to prevent accidental modification or to prevent a user from assigning a password and locking out all other users. There is no performance degradation in opening the data sets for reading.
- **Read Protection.** The password must be supplied to open the data set for reading.

## **Authorizations**

Special authorizations in the User Attribute Data Set are required for the use of some TSO facilities. Specific authorization is required for:

- Submission of jobs for execution in the background.
- Use of system operator commands from the terminal.
- Use of commands to modify the User Attribute Data Set itself.

The User Attributes Data Set should be password-protected, to prevent assignment of these authorizations by anyone other than the system administrator or his designate.

## **Capabilities of the TSO Command Language**

The TSO command language serves two separate, but related, purposes:

- It gives the terminal user a simple means to request the system to perform work.
- It gives system personnel a framework for applications.

Functions available through the commands supplied with the system include:

- Data set management.
- Program development.
- Program execution.
- System control.

The following sections describe these capabilities and are followed by a description of the applications available as IBM Program Products. Installation management has complete control over which functions are available to each terminal user.

**Data Set Management:** The TSO command language includes commands to enter, store, edit, and retrieve data sets consisting of text, data, or source programs. Essentially, the commands give the terminal user the data set management functions of the operating system. Through the use of default values and data set naming conventions, the commands can be simple enough for the non-sophisticated user.

Data from the terminal goes into standard operating system sequential or partitioned data sets. Conventions for immediate correction of keying errors are available for each terminal device type. At a 2741 Communications Terminal, for instance, the user can just backspace over an error and type in the correct characters.

At the user's option, the system will assign a number to each line of data as it is entered. Later, the user can retrieve and edit the line by referring to this line number. The user can also retrieve a line by specifying a string of characters contained in the line, and having the system scan the data set for it.

**Program Development:** TSO offers convenient facilities for program development. The programmer can use the data-handling facilities to create source programs and to have them syntax-checked line-by-line as he enters them. Any operating system language processor can be invoked from the terminal. Some language facilities and translators designed especially for the terminal environment are discussed under "IBM Program Products."

Compiler diagnostic messages and listings can be directed to the terminal, allowing the programmer to correct errors immediately and recompile the program. Once the program compiles successfully, it can be tested conversationally. The programmer can start and stop execution from the terminal, inspect and modify main storage and register contents, trace and control the program flow.

Because of background-foreground compatibility, programs produced at the terminal can be executed in either environment. Programs in the foreground can use the sequential access methods (BSAM and QSAM) to direct I/O to the terminal. In the background, the same unmodified programs can address a data set or unit record device.

**Program Execution:** Programs can be invoked at the terminal in several ways. Any load module can be established as a command and executed simply by keying in the program name at the terminal. Load modules not defined as commands can be invoked in the foreground with the CALL command. If a program uses data sets, a command procedure can be used to allocate them. Entering the one-word procedure name can allocate the data sets, invoke and start the program, and free the data sets again on program termination. Whenever a program in the foreground terminates with an error condition, the testing facilities can be used to determine the nature of the error.

The terminal user can also submit jobs to the background job stream. Commands similar to those used for the Conversational Remote Job Entry (CRJE) facility are used to create job control language describing the job, and to submit it to the batch job stream. The user can request notification of job completion at his terminal, and can have job output directed either to his terminal or to a device at the computer site.

**System Control:** Certain users can be authorized to use commands for controlling system operation. With the proper authorization, a user at a remote terminal can use standard operator commands such as DISPLAY and MODIFY to control the time-sharing portion of the system.

A separate control facility (ACCOUNT) allows an authorized terminal user to establish and maintain the profile of each system user. Using special commands from his terminal, he can define or modify user passwords, account numbers, and procedure names, and control authorizations and restrictions for each user.

## **IBM Program Products**

The command language is designed so that new commands and applications can be easily integrated into it. Applications available from IBM as Program Products look the same as other commands to terminal users.

The IBM Program Products available for TSO systems are introduced briefly in the following paragraphs. Each is discussed more fully in later chapters of this manual.

### **Problem Solving**

Three language processors specially designed for mathematical problem solving by users who are not necessarily professional programmers are available. Two are part of the Interactive Terminal Facility (ITF). The third is Code and Go FORTRAN, which is discussed with the other FORTRAN Program Products in the next section.

ITF: BASIC is a simple, algebra-like language easily learned by anyone familiar with mathematical notation.

ITF: PL/I is a subset of full PL/I that provides a powerful conversational language that is easy to learn and use. Because of its relationship to full PL/I, it is an excellent vehicle for teaching. ITF: PL/I can be executed line-by-line as it is entered, or collected into procedures and subroutines for later execution. Errors in either ITF: BASIC or ITF: PL/I can be detected as soon as the statement is entered and can be corrected immediately.

### **Programming**

Program Products to aid the users of several programming languages are available with TSO. There are three types of products:

- Compilers.
- Libraries to support the compilers and object programs.
- Prompters.

The compilers can be used in either background or foreground environments. In the foreground, they provide diagnostics and listings formatted for the terminal. For instance, diagnostic messages can optionally refer to source errors by the line number assigned by the EDIT command. With the line number, the user can retrieve and correct the statement without having a complete listing displayed at the terminal.

Prompters are initialization routines that allow the user to invoke a compiler with a single command, such as FORT or RUN. The prompter handles all data set allocations and sets processor options. If the user omits necessary information, such as the name of the source program to be processed, the prompter requests the information with a terminal message.

The following paragraphs introduce the Program Products available for each programming language. The chapter "Programming at the Terminal" discusses these products in greater detail and shows how other operating system processors can be used from the terminal.

**FORTTRAN:** There are four Program Products for FORTRAN programmers: two language processors, a library for use with either processor, and a prompter.

Code and Go FORTRAN is a quick-response, high-performance processor to meet the needs of both the problem-solver, who writes, debugs, and executes relatively short conversational programs, and the production programmer, who debugs components of a large program online before running the program through a batch compiler. The Code and Go FORTRAN processor incorporates a prompter routine.

FORTTRAN IV (G1) is an extended version of FORTRAN IV (G). It provides the ability to store permanent object programs, and produces source and object listings and storage maps. The TSO FORTRAN Prompter Program Product is available to invoke this processor.

The FORTRAN IV Library (Mod 1) is an extension of the FORTRAN IV library for use with either FORTRAN IV (G1) or Code and Go FORTRAN. It supports new features of these processors, such as a list-directed input/output facility, ASCII data set conversion, and PAUSE and STOP statements for the terminal.

**COBOL:** A language processor and prompter are available for COBOL programmers: the American National Standard Full COBOL Version 3 compiler, and the TSO COBOL Prompter.

**PL/I:** Two language processors and two supporting libraries are available for PL/I programmers: the PL/I Optimizing Compiler, the OS PL/I Checkout Compiler, and the PL/I Resident Library and the PL/I Transient Library. The compilers incorporate a prompting routine.

**Assembler Language:** The TSO Assembler Prompter is available to invoke the Assembler (F). The Assembler (F) is not a Program Product.

### **Text and Data Handling**

The TSO Data Utilities: COPY, LIST, MERGE, FORMAT Program Product provides four commands to manipulate data sets, and to format text for output either at the terminal or on a high-speed printer.

## Command Language Facilities

TSO terminal users define their work in the TSO command language. A command can be thought of as a request from the terminal user for the system to perform a particular function. This chapter describes and gives examples of the facilities available through the command language. There are commands for elementary functions such as entering, editing, and retrieving data; remote job entry; mathematical calculation; and program development and testing in several programming languages. These important functions are the base on which the installation's own terminal-oriented applications and systems are developed.

To make the example more meaningful, some TSO conventions for command syntax, entry format, data management, and terminal operation are presented first. Many of these conventions can be redefined at the option of the installation, or in some cases, at the convenience of the individual user.

### Conventions at the Terminal

The command is the means by which work is defined at the terminal. The first word of a command is always the **command name**. It is used by the system to select a command processor (a problem program) from the system command library or a user command library. Any further information in the input line, the command operands, is passed to the command processor in a parameter list. Operands are separated, or delimited, by either blank spaces or commas. A few commands require that groups of related operands be enclosed in parentheses.

Most operands are optional. If an optional operand is not entered with the command, the system assumes the default values and proceeds as if the user had entered that value. If the missing operand is not one that can be defaulted, for instance, a data set name, the system prompts the user for it with a message such as "ENTER DATA SET NAME". When all the operands have been either entered or defaulted, the command processor proceeds to perform the desired function. Some of the command processors, such as EDIT, accept, interpret, and perform subcommands, which follow the same syntactic rules as the general commands.

### Logging On

To establish a connection with the system, the user activates his terminal, dials the computer, if necessary, and enters the LOGON command. He must always supply his **user identification** as an operand of the LOGON command; if he does not supply it, a prompting message is issued. Up to three additional levels of identification may be needed, depending on the accounting methods and security procedures used by his installation.

The installation may require users to enter a **password** with the LOGON command. Each user can have one or more passwords associated with his identification. At terminals equipped with the print inhibit feature, the system is able to suppress printing of the password as it is keyed in.

Associated with each password are one or more account numbers, and with each account number, one or more LOGON Procedure names. The LOGON Procedure contains the Job Control Language statements defining the user's terminal job, just as cataloged procedures define background jobs. For instance, the LOGON Procedure may allocate certain commonly used data sets. Whenever there is only one account number or procedure name, the system selects it by default, and the user is not required to enter it. Figure 1 shows a simple identification scheme, with just one value for each of the possible levels of identification.

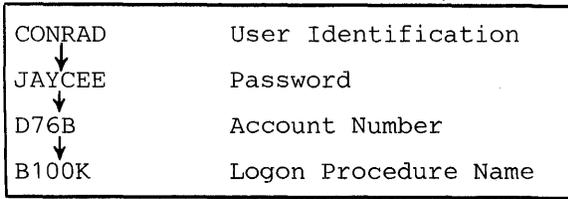


Figure 1. Simple Identification Scheme

To log on, the user defined in Figure 1 would enter:

```
LOGON CONRAD/JAYCEE
```

The system will assume "D76B" as the account number and "B100K" as the LOGON Procedure name, since no others have been defined for this user.

A user who has several account numbers and LOGON Procedures can have a hierarchy of identification values, like that shown in Figure 2. This user could still log on with the command shown above, since "D76B" and "B100K" are the only account number and LOGON Procedure name associated with the password "JAYCEE". However, to use the "X100K" LOGON Procedure, this user's LOGON Procedure, this user's LOGON command would be:

```
LOGON CONRAD/JOE#1 ACCT(D58B) PROC(X100K)
```

Both the account number and the procedure name must be included, since a choice exists for both. The information scheme for each user is defined and maintained in the User Attributes Data Set with the ACCOUNT command, by the system manager or a user authorized to do so.

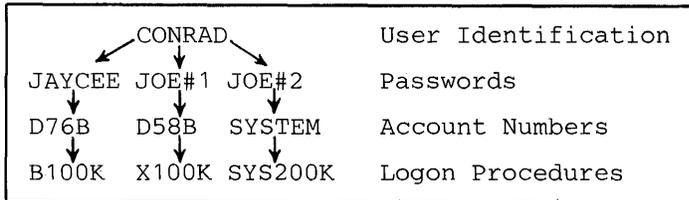


Figure 2. User Identification Hierarchy

The system acknowledges that the LOGON has been accepted when it has checked the identification supplied and has determined that the resources requested in the LOGON Procedure are available, and the user can begin his work.

### Input Editing

Some system editing is provided for every line of input from the terminal. Lowercase alphabetic characters (from terminals that have them) are translated to uppercase, except for certain text-handling applications. Each user can define his own character-delete and line-delete characters for correcting any keying errors in input lines. There are default character-delete and line-delete characters for the typewriter-like terminals (the cursor controls can be used on the 2260 and 2265 Display Stations). If a user defines the quotation mark as his character-delete character, and percent sign as his line-delete character, then enters the line:

```
etoain%aBcC"deee"
```

it is received by the system as

```
ABCDE
```

Users whose terminals have backspace and attention keys may define those keys as their character-delete and line-delete characters.

### **Entry Modes**

Immediately after LOGON, the system is ready to accept any command in the command libraries. The terminal is said to be in command mode. Some commands place the terminal in other entry modes: EDIT, for instance, has an **input mode**, that accepts successive lines of input for a data set; and an **edit mode**, that accepts EDIT subcommands. Other commands, such as TEST, ACCOUNT, OUTPUT, and OPERATOR also define special purpose modes.

### **The Attention Key**

The attention key can be used to transfer from one mode to another, or to interrupt a program or command processor during execution. Any command in process can be cancelled by hitting the attention key and entering a new command. A user program can be interrupted with the attention key to transfer to the test mode for debugging activity, then the program can be restarted.

Assembler language user programs can define **attention exit routines** with the STAX macro instruction. Control will be passed to such a routine when an attention is entered.

An important function of the attention feature is to prevent the user from being “locked out” of the system while a long-running program executes, or while voluminous output is displayed at the terminal. For terminals without attention keys, the attention feature can be simulated. The user can specify a string of characters, such as “STP”, that is to be interpreted as an attention. The system can be instructed to interrupt any long-running program or terminal output periodically to accept either the simulated attention character string, or a digit to specify the level of attention exit or a null line to continue processing.

### **Data Set Naming Conventions**

Unless requested not to, the system appends two qualifiers to a simple data set name specified by a user: a descriptive qualifier and user identification qualifier. The user identification qualifier is the same as the user identification specified with the LOGON command and is appended as the left-most qualifier of data set names that follow the TSO naming conventions.

The descriptive qualifier is placed at the right of the user entered data set name. It tells the system what kind of data is recorded in the data set and for what purposes it can be used. For instance, the qualifier for a data set containing COBOL source statements is COBOL; for a load module, LOAD. Whenever possible, the system determines the appropriate descriptive qualifier from the command referring to the data set, and the user need not enter it as part of the name. In some cases, the user must supply it, as part of the data set name entered with the command, or in response to a prompting message.

The user never needs to enter his identification qualifier; it is always known to the system. Figure 3 is an example of a series of commands to enter and compile a source program (using the TSO FORTRAN Prompter and FORTRAN IV (G1) Program Products); linkage edit the object program; to display the compiler listing at the terminal; and finally, to delete all the data sets involved.

| Input Data Sets   | Command & Operands | Output Data Sets                    | Comments                             |
|---|--------------------|-------------------------------------|--------------------------------------|
| none  | edit prg1 new fort | CONRAD.PRG1.FORT                    | FORTRAN source program.              |
| CONRAD.PRG1.FORT  | fort prg1 print    | CONRAD.PRG1.OBJ<br>CONRAD.PRG1.LIST | Object program.<br>Compiler listing. |
| CONRAD.PRG1.OBJ   | link prg1          | CONRAD.PRG1.LOAD<br>(TEMPNAME)      | Load module, in a one-member PDS.    |
| CONRAD.PRG1.LIST  | list prg1.list     | Display at terminal                 | Compiler listing                     |
| CONRAD.PRG1.FORT<br>CONRAD.PRG1.OBJ<br>CONRAD.PRG1.LIST<br>CONRAD.PRG1.LOAD<br>(TEMPNAME) | delete prg1.*      |                                     | All the data sets are deleted.       |

Figure 3. Example of Data Set Naming Conventions

In the EDIT, FORT, and LINK commands, the user supplied only the simple data set name, "PRG1". The system assigns the descriptive qualifiers implied by the commands and their operands. With the LIST command, the user supplied the descriptive qualifier, since he might want have wanted to display either CONRAD.PRG1.FORT or CONRAD.PRG1.LIST. In the DELETE command, the user entered an asterisk in the descriptive qualifier field, to tell the system to delete any data sets with the identification qualifier CONRAD and the simple name PRG1.

To refer to a data set that does not follow the naming conventions, or that has an identification qualifier different from the one specified at LOGON, the user encloses the fully qualified data set name in apostrophes. The system does not append any additional qualifiers in this case, and uses the name "as is," except for translation to uppercase, to search the catalog. Using this technique, several users may refer to a data set with the shared attribute at the same time.

## Data Entry

The EDIT command is used to enter information into the system. Because almost every system application will use some of the editing facilities, an overview of the command is presented here. Later sections will demonstrate some of the particular uses of the command.

### Creating Data Sets

The EDIT command processor creates or modifies data sets with sequential organization, including members of partitioned data sets. The data sets contain only printable characters in EBCDIC representation. A data set name is entered with the EDIT command. If the user specifies the data set is old, EDIT opens it for modifications. If it is a new data set, EDIT invokes the dynamic allocation routines to create it. The data set attributes, such as blocksize and record length, can be specified by the user, or defaulted to standard values. For data sets containing source-language programs, the standard attributes are determined by the compiler to be used.

One input line from the terminal normally becomes one record in the data set. Because of this equivalency between records and lines at the terminal, data sets created by EDIT are called **line data sets**. On request, EDIT prefixes a line number to each record of the data set as it is entered.

## **Entry Modes for EDIT**

Depending on the type of work the user is doing with his data set, he uses one of two entry modes provided by EDIT (some other modes specifically for particular programming languages are discussed later). The input mode allows rapid entry of successive lines of input for the data set. The edit mode allows subcommands to be entered to modify, insert, or delete lines.

### **Input Mode**

In input mode, the user enters successive lines of input. The lines are normally appended at the end of the data set, although the user can request they be inserted at some other point. The only subcommand recognized in the input mode is the null line (hitting the return key with no preceding characters), which requests transfer to the edit mode.

Services available in the input mode include translation of lowercase letters to uppercase, translation of tab characters to a series of blanks, and interpretation of the character-delete and line-delete characters. If line numbers are being assigned to each line, the user may request each new number be typed out by the system at the beginning of each input line. If line numbers are not being assigned, the user can request prompting for each new line by an underscore. If no prompting is requested, lines are entered one after another, with no intervening response from the system. Programming language syntax checkers can be requested to process input lines as they are entered.

### **Edit Mode**

In edit mode, the user enters subcommands to point to particular records of the data set, to modify or renumber records, to add and delete records, to control editing of input, or to compile and execute a program.

Whenever the terminal is in edit mode, the user is considered to be positioned at a particular record, or line, of the data set. The EDIT command processor maintains a current line pointer to keep track of the user's position. In general, the **current line pointer**, which can be referred to in subcommands by an asterisk (\*), is positioned at the last line referred to, entered, changed, or printed. Using the subcommands provided, the user can move the current line pointer to any record in the data set.

For line-numbered data sets, specifying a line number as an operand of a subcommand moves the pointer to that record before the action requested by the subcommand is carried out. This method of operation is called **line number editing**.

Another method of repositioning the current line pointer is called **context editing**. A set of subcommands is provided to reposition the current line pointer without reference to line numbers. The user can move the pointer up or down a specified number of lines, or request the system to find a line with a particular series of characters in it, and move the pointer to it.

### **Modifying Data Sets**

The most common use of the EDIT command for existing data sets is the addition, deletion, or modification of records. The INSERT and DELETE subcommands handle single or multiple record insertions and deletions. The CHANGE subcommand allows the user to replace one character string with another, not necessarily of the same length.

## Data Set Management Commands

To allow the user to manage his data stored on auxiliary storage devices, a set of data set utility commands is included in the TSO command language. All user data is kept in standard operating system data set, and as a default, data sets used by foreground programs are entered in the system catalog, reducing the amount of information the user must supply about the data set from the terminal when he refers to it.

The LISTCAT and LISTDS commands retrieve information from the system catalog for the user. He can find out what data sets are currently allocated to him, and what the attributes of the data sets are. The RENAME command can assign a new data set name to an existing data set, or add an alias name to a partitioned data set member. The DELETE command removes a data set from the catalog, and frees the auxiliary storage space it occupies.

The PROTECT command is the facility to assign password protection to data sets. Protection can be assigned for read access and for write and delete access. Multiple passwords can be assigned to a single data set.

The ALLOCATE and FREE commands invoke the dynamic data set allocation routines from the terminal. A user who wants to run a program that requires one or more data sets not currently allocated to his foreground job enters ALLOCATE commands to have the data sets assigned. The FREE command is used to release the data sets assigned by ALLOCATE. The ALLOCATE command can also be used to find data sets not in the system catalog, and to control the size of new data sets and the volumes to which they are assigned.

The ATTRIB command can be used to build a list of data set attributes. These attributes are specified by the operands of the command. The operands are similar to the DCB parameters of the JCL DD statement. The attributes in an attribute list can be assigned to data sets being allocated by the ALLOCATE command.

## TSO Data Utilities

The TSO Utilities Program Product is available to augment the data entry and data set management commands by providing a text-formatting capability and data set utilities for terminal users. The product provides four commands:

- FORMAT, to format textual information into pages.
- LIST, to display all or part of a data set at the terminal.
- COPY, to copy a data set.
- MERGE, to merge all or part of one data set into another.

The FORMAT and MERGE commands can also be used as subcommands of EDIT (EDIT incorporates a less powerful listing capability). The COPY and MERGE commands can be used for access to ASCII tape data sets. See the publication *IBM System/360: Planning for the Use of Information Interchange Standards*, GC28-6756, for details.

### Text-Handling

The EDIT, FORMAT, and LIST commands provide a facility for the entry, editing, storage, and output of text. With the EDIT command, the terminal user creates a data set with the type qualifier TEXT, and enters the material line-by-line. If his terminal has both uppercase and lowercase letters, the material will not be translated to uppercase letters, but will be saved just as entered. The user can specify what tab settings he wants to use, and the system will convert tabs in the material into strings of blanks of the proper length. The use of line numbers is optional.

The user formats the data set by inserting format control records into it. A format control record is entered as a separate line in the data set, starting with a period in the first position, followed by a control word (or a two-character abbreviation). The EDIT processor does not interpret the controls; they are retained in the data set for interpretation later by the FORMAT processor. The controls allow the user to:

- Print a heading on each page.
- Center lines of text between margins.
- Control the amount of space for all four margins on the page.
- Control line spacing.
- Justify left and right margins of the text.
- Number pages of output consecutively.
- Halt printing when desired.
- Print multiple copies of selected pages.

The FORMAT processor scans the data set for the format controls and inserts blanks, carrier return characters, headings, and page numbers as needed. At the user's option, the output can be formatted for a terminal or saved in a data set for deferred printing, either on the terminal (with the LIST command) or on a high-speed printer. Either an all-capitals or an uppercase and lowercase print chain can be used on the printer.

### **Data Set Manipulation**

The COPY, LIST, and MERGE commands allow the terminal user to move information between data sets and to display sets at the terminal.

The COPY command will duplicate sequential or partitioned data sets or a member of a partitioned data set. While doing so, it can resequence or change the record length, blocksize, or record format as requested. The MERGE command will copy all or part of one data set or member into another data set and will resequence the record numbers in the receiving data set if requested. Both these data commands will process tape data sets in ASCII format. Tape devices must be allocated to a user in his LOGON procedure.

The LIST command displays all or part of a data set at the terminal. The user can request that fields within records be rearranged for output, and line numbers can be suppressed or included.

## **Compiling and Executing Programs**

A variety of commands are provided to give the user control over program compilation and execution. The form of the program determines command selection. For those language processors that are supported by a prompter Program Product or that incorporate a prompter, the terminal user requests a compilation of a source program with a single command. The prompter performs the following functions:

- Requests any necessary operands with messages to the terminal.
- Sets other compiler options to default values suitable for the terminal environment.
- Dynamically allocates the data sets needed by the compiler.
- Invokes the compiler.

For instance, if an installation has the TSO COBOL Prompter and the American National Standard Full COBOL Version 3 processor Program Products, the user can enter the COBOL command to compile his program and produce an object module. The LOADGO command can then be used to call the OS Loader to bring the program into main storage for execution, or the LINK command can be used to call the Linkage Editor to create a permanent load module.

During program development, when a programmer is repeatedly compiling and testing a program, he can use the RUN command to invoke it. RUN first calls the appropriate prompter and compiler, and then the OS Loader (except for the PL/I Checkout Compiler). It provides a compile-load-go sequence with a single command. RUN can be used as a command, or as a subcommand of EDIT. Figure 4 is a summary of the commands for executing programs. The chapter "Programming at the Terminal" has examples of the use of these commands. You must use RUN with ITF: PL/I, GOFORT, and ITF: BASIC.

| Form of input: | Form of output: | OBJECT MODULE               | LOAD MODULE | EXECUTING PROGRAM |
|----------------|-----------------|-----------------------------|-------------|-------------------|
| SOURCE PROGRAM |                 | COBOL<br>FORT<br>ASM<br>PLI | --          | RUN               |
| OBJECT MODULE  |                 | --                          | LINK        | LOADGO            |
| LOAD MODULE    |                 | --                          | --          | CALL              |

Figure 4. Program Control Commands

Any load module, including language processors for which there are no prompters, can be invoked with the CALL command. For instance, the FORT command provided by the TSO FORTRAN Prompter Program Product invokes the FORTRAN IV (G1) compiler. If a programmer wants to use the FORTRAN (H) processor for a particular compilation, he can enter the command:

```
CALL 'SYS1.LINKLIB(IEKAA00)' 'MAP,OPT=1'
```

The compiler is loaded into the foreground region and given control. The options are passed to it as though they had been specified in the PARM field of an EXEC statement in Job Control Language. It is necessary for the user to allocate data sets for the compiler's use before entering the CALL command. A series of ALLOCATE commands can be defined in a command procedure, so that they need not be entered every time a compiler is used. An example of such a procedure is included in the chapter "Programming at the Terminal."

The TEST command can also be used to invoke a user program, and to control its execution. Before passing control to the program, TEST allows the user to establish initial values to be passed to the program as test data, and to set up breakpoints where execution is to be interrupted for displays and other debugging activity.

Breakpoints are established with the AT subcommand in test terminal mode. AT specifies a symbolic or absolute address in the program where execution is to be interrupted. The action to be taken at the point of interruption, such as listing or modifying storage and register contents, can be specified in a pre-stored string of TEST subcommands, or entered through the terminal at the time of interruption. Main storage contents can be displayed at the terminal or stored in a data set for deferred printing. TEST subcommands allow the programmer to load additional programs into storage, to delete or replace programs in storage, to issue GETMAIN and FREEMAIN as subcommands from the terminal, to define the location and attributes of symbols, and to start and stop program execution.

## Remote Job Entry

The command language includes the **SUBMIT**, **STATUS**, **OUTPUT** and **CANCEL** commands to handle submission of jobs for execution in the background. These commands have the same format as the commands available with the Conversational Remote Job Entry (CRJE) facility of the operating system.

To have a job executed in the background, the user places the job control statements defining the job in a data set. By convention the jobname is the one-to-seven character user identification, plus a single character to provide uniqueness. The user then enters a **SUBMIT** command, including the name of the data set as an operand. **SUBMIT** will generate a standard jobname and a **JOB** statement if one is not included in the job definition. One data set can contain more than one job definition. Any time after entering the **SUBMIT** command, the user can inquire about the status of the job. The **STATUS** command returns information such as whether the job is waiting for resources, is executing, or is completed. The job can be terminated with the **CANCEL** command.

A new keyword has been defined for the **JOB** statement to allow automatic notification of the user when the job is completed. By coding **NOTIFY=** with his user identification, the user requests a message to his terminal when the job completes. The message is saved until he enters a **LISTBC** command. The **OUTPUT** command allows the user to display job output (**SYSOUT**) at his terminal, to save it in a data set, or to delete it.

## System Control

Two facilities are provided for the installation manager or system programmer to control operation of the system from his terminal. The **ACCOUNT** command adds, changes, or deletes entries in the User Attributes Data Set, which is the list of all authorized users of the system, together with the characteristics defining their profiles. The **OPERATOR** command places a terminal in a special mode allowing entry of commands normally available only at the system console. Use of either of these facilities requires special authorization. Users with such authorization are called **control users**.

### User Authorization

When a control user enters an **ACCOUNT** command, his terminal is placed in account mode. With subcommands, the control user defines each user to the system, specifying his identification, passwords, account numbers, and **LOGON** Procedure names. This information is placed in the User Attribute Data Set, along with indications of any special authorizations the user may have, such as permission to use the **ACCOUNT** or Remote Job Entry facilities, and a limit on the region size he may request. This information will be retrieved whenever the user logs on, to verify his authority to use the system, and to define his foreground job.

### System Operation

By entering the **OPERATOR** command, a control user has access to the system operator commands **MODIFY**, **DISPLAY**, **MONITOR**, **CANCEL**, and **STOPMN**. The commands have the same format and effect on the TSO system as if they were entered through the operator's console, as specified in the publication **IBM System/360 Operating System: Operator's Reference**, GC28-6691.

## Command Procedures

A command procedure is a data set containing a list of TSO commands and subcommands. The data set name is entered as an operand of the EXEC command, and the commands are executed, one-by-one in the order in which they appear in the procedure. When one command or subcommand is completed, the next is read from the procedure and processed as though it had been entered from the terminal. The commands can be typed out at the terminal as they are executed, or the user can suppress the listing with an operand of the EXEC command.

The EXEC command can also be invoked implicitly if the procedure is a member of the command procedure library. The member name of the command procedure can be entered as a command name. When the name is not found in the command libraries, the system assumes it is in the command procedure library.

**Operand Substitution:** Symbolic operands, starting with an ampersand ( & ), can be placed in commands and subcommands within command procedures. Values for these operands are supplied in the EXEC command invoking the procedure. A procedure (PROC) statement at the beginning of the procedure specifies how many positional operands will be supplied, and what keyword operands may be present. Default values for symbolic operands can be specified in the PROC statement.

**Conditional Statements:** The WHEN statement tests the return code from any command or program invoked during a procedure. A condition is stated with relational operators such as GT or LT (“greater than” or “less than”). If the condition is satisfied, the command in the WHEN statement is executed. If it is not satisfied, the command following the WHEN statement is executed. The command in the WHEN statement can itself be an EXEC command, invoking another command procedure.

Figure 5 shows a command procedure with a symbolic operand and a conditional execution statement. The procedure has commands to linkage edit an object program (LINK), and bring it into main storage for testing (TEST). The symbolic operand “PROGNAM” is defined in the procedure statement beginning the procedure. If the member name of this procedure in the procedure library is “LDTST”, the command to invoke it is:

```
LDTST MYPROG
```

The commands in the procedure are then executed in order, with the substitution of “MYPROG” for “& PROGNAM” wherever it appears. A period with the symbolic operand specifies concatenation with the adjacent field. The value for “& PROGRAM.” is concatenated to “OBJ” the second “.”, and the LINK command as executed is:

```
LINK MYPROG.OBJ TEST MAP XREF
```

The WHEN statement causes the return code from the linkage editor to be tested. If it is zero, indicating no errors, execution proceeds with the TEST command. If the return code is greater than zero, indicating linkage editing errors, the procedure is terminated, and the next command can be entered from the terminal.

|      |                             |
|------|-----------------------------|
| PROC | 1 PROGNAM                   |
| LINK | &PROGNAM..OBJ TEST MAP XREF |
| WHEN | SYSRC(GT 0) END             |
| TEST | &PROGNAM..LOAD(TEMPNAME)    |
| END  |                             |

Figure 5. A Command Procedure

## Other Commands

Several other commands are provided to allow the user to control the terminal environment and to aid him in using the command system.

The **TERMINAL** and **PROFILE** commands are used to tailor the data entry conventions to the terminal type and user's preference. **TERMINAL** allows him to specify the character string to be used to simulate an attention interruption if his terminal does not have an attention key, and to specify how often he is to be given an opportunity to simulate an interruption during long-running execution or output sequences. The **PROFILE** command is used to specify the character-delete and line-delete characters, and other user options such as whether he wants prompting messages suppressed.

The **HELP** command provides the user with reference information on command and subcommand syntax, function, and usage. For example, if a user has forgotten the function of the **DELETE** command, he can enter:

```
HELP DELETE FUNCTION
```

The **HELP** command will return information explaining the function of the **DELETE** command:

```
THE DELETE COMMAND IS USED TO DELETE A SEQUENTIAL DATA SET OR A MEMBER OF A PARTITIONED DATA SET.
```

Requesting this information through the terminal is faster than searching for it in a printed manual.

The **SEND** command is used to send a message to the system operator or to another user. The sender must know the user identifications of other users to whom he directs messages. Messages are displayed immediately at the receiver's terminal unless the receiver has requested that messages be suppressed or unless he is not logged on. Messages not sent immediately are saved, and are transmitted if requested, when the addressee next logs on.

## Programming at the Terminal

The time sharing environment is especially well-suited to program development. The advantage of programming at a time sharing terminal is the reduction of job turn-around delays. The programmer can profitably devote himself to one project at a time -- he does not need other projects to work on while waiting for results from a batch computing facility. TSO provides services for terminal users at each step in program development: coding, compiling or assembling, testing, implementation, documentation, and program maintenance.

Any compiler or assembler designed to run under the operating system can be invoked from a TSO terminal. Compilers can be executed in the foreground, or, via the SUBMIT command, in the background. Command language prompters are either incorporated in or separately available for several of the language processors. The TSO prompters, all IBM Program Products, provide specific commands to invoke the associated processors, and perform the following functions:

- Requests the user to enter necessary information such as the name of his source program.
- Allocates data sets required by the processor and frees them on completion.
- Sets any compiler options specified by the user and sets default values for those options the user omits.

Prompters are available for American National Standard COBOL Version 3, FORTRAN IV (G1), and Assembler (F). Prompters are incorporated in the PL/I Optimizing Compiler, the OS PL/I Checkout Compiler, and the problem-solving language processors. Each of the processors accepts a TERM option, a request for special formatting of diagnostic and listings for the terminal, and a NUM option, to control the use of EDIT line numbers in error messages. Syntax checking of source programs is provided for PL/I (F), FORTRAN IV levels (E), (G1), (G), and (H), and the problem-solving languages. The test mode gives the user real-time control over program execution for debugging. Similar facilities for ITF: PL/I, ITF: BASIC, and Code and Go FORTRAN and the PL/I Checkout Compiler are discussed in the next chapter, "Problem Solving."

Either the loader or the linkage editor can be invoked from the terminal. Users authorized to do so can add load modules to the system command library, where they will be available as commands to all system users. Any user can add programs to his own command library. Programs written in any language can be defined as command processors, but only assembler language has the facilities to make use of the command service routines such as input scanning and prompting.

Because of foreground-background compatibility, production programs that will eventually be run in the background environment can be written and tested from the terminal. I/O that goes to the terminal in the foreground can be re-directed to a sequential data set in the background. No recompilation is necessary.

The following sections describe the special terminal support provided for COBOL, FORTRAN, PL/I, and Assembler language programmers. Language processors for which no specific terminal support is provided can also be invoked in the foreground. See "Other Compilers" in this chapter for an example showing how to invoke the PL/I (F) compiler.

## COBOL

TSO provides the COBOL programmer with facilities for entering, compiling, and testing programs from his terminal. The programmer can use the COBOL command for compiling his program with the following IBM Program Products:

- TSO COBOL Prompter.
- American National Standard COBOL Version 3 Compiler.

The user can also invoke the COBOL (E) compiler, either in the foreground or the background, but only the American National Standard COBOL compiler provides listings and diagnostic messages specifically formatted for a remote terminal environment. The full American National Standard COBOL is provided, with the IBM extensions to the language as defined in the publication, **IBM System/360 Operating System: American National Standard COBOL, GC28-6396**.

### Entering the Source Program

The COBOL programmer uses the TSO EDIT command to create or modify his source program. With the EDIT command, he enters operands to name the data set containing the program, and identify it as an old program to be modified or a new program.

With the terminal in input mode, the user enters successive lines of the program. The system accepts each line when he hits the return key of the terminal, and types out the line number of the next line. This line number becomes the sequence field of the COBOL statement in columns 1-6, and in addition is used in place of the compiler-generated "card number" in program listings and diagnostic messages. Automatic line numbering can be suppressed, if desired, by an operand of the EDIT command.

After the line number is typed out, the terminal is logically positioned at the continuation column (column 7) of the COBOL statement. The user can space or tab to Area A (column 8) or Area B (column 12) of the statement. These logical tab settings are automatically set by the EDIT program whenever a COBOL program is being processed. EDIT converts the tab characters to the necessary number of blanks to format the statement correctly. The number of blanks generated is independent of the physical tab settings at the terminal. The user can, if he wishes, override the standard settings by specifying his own with an EDIT subcommand.

Figure 6 shows an EDIT command to create a new data set, and some lines from the user's COBOL program as he enters them. The five-digit line numbers are generated by EDIT. A high-order zero is appended to form the six-digit COBOL sequence field. In this example, the backspace key is used as the character-delete character, and the attention key is the line-delete character. In lines 00020 and 00050 the user backspaces over keying errors to correct them. Line 00070 is cancelled with the attention key and re-entered. The lowercase alphabetic characters are automatically translated to uppercase by the EDIT processor.

```
edit query.cobol new
INPUT
00010 identification division.
00020 progrmm-id. query.
00030 remarks. sample inquiry program.
00040 environment division.
00050 configuratiøn section.
00060 source-computer. ibm-360-i65.
00070 input-output!D
00070 object-computer. ibm-360-i65.
00080 input-output section.
00090
```

Figure 6. Entering a COBOL Program

## Compiling a COBOL Program

The American National Standard COBOL Compiler is invoked with the COBOL command. The only required operand is the name of the data set containing the source program to be compiled. However, any of the compiler options (except DECK) can be entered with the command as operands. These options are documented in the publication **IBM System/360 Operating System: American National Standard COBOL Programmer's Guide, GC28-6399**.

Two compiler options are available: TERM and NOPRINT. The TERM option orders the compiler to issue progress messages to the terminal as it processes the source program, for instance, "ANS COBOL IN PROGRESS," and to issue diagnostic messages formatted for the terminal. An error or warning message directed to the terminal includes the line number of the source statement in error, and the compiler error message. Using edit mode subcommands, the programmer can retrieve the statement by line number, and correct the error.

The PRINT/NOPRINT option, available only in the foreground, allows the programmer to choose whether the program listing is to be placed in a data set, displayed at the terminal, or suppressed. When developing a program from the terminal, it is not normally necessary to have the complete listing generated and displayed, since the error and diagnostic messages are extracted and displayed through the TERM option. NOPRINT is the default value, and suppresses the listing. When source program errors have been corrected and the program is compiled a final time, the programmer specifies PRINT to generate the listing, which may be displayed at the terminal, or saved in a data set for deferred printing, either at the terminal or on a high-speed printer. The contents of the listing are controlled by the other of the listing are controlled by the other options such as SOURCE, PMAP, and XREF.

Figure 7 is an example of a COBOL compilation, correction of a source program error, and re-compilation. The PRINT operand of the second COBOL command causes a listing to be generated and saved in the data set QUERY.LIST.

```
cobol query
*STATISTICS*   SOURCE RECORDS = 59 DATA DIVISION STATEMENTS = 15
                PROCEDURE DIVISION STATEMENTS = 18
*OPTIONS IN EFFECT*   SIZE = 81920 BUF = 2768 LINECNT = 57
*OPTIONS IN EFFECT*   SPACE1, FLAGW, NOSEQ, NOSOURCE, NODMAP
*OPTIONS IN EFFECT*   NOPMAP, NOCLIST, NOSUPMAP, NOXREF, NOSXREF
*OPTIONS IN EFFECT*   LOAD, NODECK, APOST, NOTRUNC, NOFLOW, TERM
*OPTIONS IN EFFECT*   NUM, NOBATCH, NONAME, COMPILE=01, NOSTATE
001 COMPILATION ERRORS. HIGHEST SEVERITY E
000290 IKF3001I-E INV-KY-READ NOT DEFINED. STATEMENT DISCARDED.
READY

edit query.cobol
EDIT
list 290
000290 READ PARTS-FILE INVALID KEY GO TO INV-KY-READ.
change /ky/key/
save
SAVED
end
READY

cobol query source xref dmap pmap
ANS COBOL IN PROGRESS
READY
```

Figure 7. Compiling a COBOL program

## Program Execution

The object program created by the COBOL compiler can be invoked with the LOADGO command, which calls the Loader to bring the program into main storage and pass control to it. The user enters ALLOCATE commands for any data sets needed by the program before invoking it.

The object program can also be defined as, or as part of, a load module with the LINK command. As a load module, the program can be placed in a private or system program library. To define the program QUERY as a command in a private command library CONRAD.COMMANDS.LOAD, the LINK command is:

```
LINK QUERY.OBJ LOAD(COMMANDS(QUERY)) COBLIB
```

Once part of a command library, and once the private command library is concatenated to the command library (and linkage library), the program is invoked simply by entering the program name (or an alias) as a command. To invoke the program defined above, the user would type:

```
QUERY
```

The RUN subcommand of EDIT functions as a combination of the COBOL and LOADGO commands. It is especially useful during the testing phase of program development, since it can be used without leaving the edit mode. When a source program is complete, the user enters the RUN command, invoking first the compiler, then his object program. Whenever he detects an error requiring a change to the program, the programmer can immediately update his source program with EDIT subcommands, and enter another RUN subcommand.

## Interactive Programs

COBOL programs can be designed to interact with a terminal user simply by defining the terminal as a file. Programs can read input lines from the terminal, act on the information, and respond. Because the terminal is defined as a sequential utility file, the same program can be executed in the background, reading and writing to sequential data sets or devices, without recompilation.

To define the terminal as a file, the user enters ALLOCATE commands for the external names used in the name field of the ASSIGN clauses in the program. Figure 8 shows a FILE-CONTROL statement in a COBOL program, and a corresponding ALLOCATE command that assigns the file to the terminal at execution time.

The DATASET operand of the ALLOCATE command corresponds to the DSNNAME field of a DD statement. The \* is a conventional symbol for the terminal. The FILE operand corresponds to a DDNAME. In this example, any output written to ANSWER-FILE in the program will be displayed at the terminal.

The same program could be executed in the background, with a DD statement directing the output to the printer or data set.

```
SELECT ANSWER-FILE ASSIGN TO UT-S-DDOUT
allocate dataset (*) file (ddout)
```

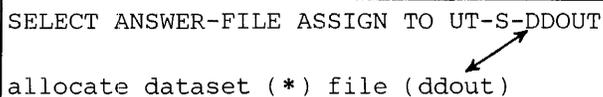


Figure 8. Defining the Terminal as a File

For programs containing ACCEPT and DISPLAY clauses, or which generate TRACE and EXHIBIT output for debugging, the SYSIN and SYSOUT files can be defined as the terminal. The ALLOCATE commands are:

```
ALLOCATE DATASET( * ) FILE(SYSIN)
ALLOCATE DATA SET( * ) FILE(SYSOUT)
```

DISPLAY output is sent to the terminal instead of system output. TRACE and EXHIBIT output is also sent to the terminal.

### A COBOL Example

Figure 9 is an example of a terminal session in which the user writes, compiles, and executes a simple inquiry program in COBOL. The program uses a part number entered from the terminal to select a record from an indexed file, and prints the information from the record out on the terminal.

At 1, the user enters an EDIT command to create a new data set that will contain the COBOL source program. The EDIT processor recognizes the descriptive qualifier "COBOL" in the data set name, and sets the record length and blocking factor to values acceptable to the ANS COBOL compiler. The tabs in the data set are set to "columns" 7, 8, 12, and 72. Since it is a new data set, the EDIT processor requests input (at 2), and types out the first line number ("00010" at 3). The EDIT processor continues to generate line numbers until 5, where the user enters a null line to request transfer to edit mode.

In the program itself, the user defines an indexed sequential file in the INPUT-OUTPUT section. This data set is permanently allocated by a DD statement in the user's LOGON procedure. No files need to be defined for the terminal in the LOGON procedure, since the ACCEPT and DISPLAY statements can be directed to the terminal with ALLOCATE commands defining SYSIN and SYSOUT.

At 7, the user saves a copy of his source program, and enters the END subcommand to terminate the EDIT processor. The COBOL command at 10 invokes the ANS COBOL compiler to process the source program.

At 13 and 14, the user assigns SYSIN and SYSOUT data sets to the terminal. This same COBOL program could be executed in the background by including SYSIN and SYSOUT DD statements in the Job Control Language.

The LOADGO command at 15 calls on the Loader to bring the object program into main storage and pass control to it. The COBLIB operand specifies that the standard COBOL library is to be used to resolve external references.

At 16, after his keyboard unlocks, the user enters a part number to test the program. The program locates the corresponding record in the indexed file, and displays the information in it at 17. The user repeats the test at 19, but with a non-existent part number. At 22, the user invokes the Linkage Editor to create a load module, and add it to his command library, a partitioned data set concatenated to the system command library. Once the member is added to his command library, the user can invoke the program by entering the program name as a command, as he does at 23.

```

1  edit query.cobol new
2  INPUT
3  00010 identification division.
   00020 program-id. query.
   00030 environment division.
   00040 configuration section.
   00050 source-computer. ibm-360-i65.
   00060 object-computer. ibm-360-i65.
   00070 input-output section.
   00080 file-control.
   00090     select parts-file assign to da-2314-i-ddparts
   00100     access is random
   00110     nominal key is key-in, record key is rec-key.
   00120 data division.
   00130 file section.
   00140 fd parts-file
   00150     block contains 5 records
   00160     record contains 80 characters
   00170     label records are standard.
   00180 01 parts-record.
   00190     02 delete-code           picture x.
   00200     02 rec-key             picture 9(7).
   00210     02 part-history        picture x(72).
   00220 working-storage section.
   00230 77 key-in                 picture 9(7).
   00240 procedure division.
   00250 begin.
   00260     open input parts-file.
   00270 para-1.
   00280     accept key-in.
   00290     read parts-file invalid key go to inv-key-read.
   00300     display rec-key, part-history.
   00310 eojb.
   00320     close parts-file. stop run.
   00330 inv-key-read.
   00340     display 'no record found'.
   00350     go to eojb.
4  00360
5  EDIT
6  save
7  SAVED
8  end
9  READY

10 cobol query so x d pm
11 ANS COBOL IN PROGRESS
12 READY

13 allocate dataset(*) file(sysin)
   READY
14 alloc da(*) f(sysout)
   READY

15 loadgo query coblib
16 6367220
17 6367220 SHIM CLIP    24 ON HAND    10 ORDER POINT SUPPLIER 17688
   READY

18 loadgo query coblib
19 9999999
20 NO RECORD FOUND
21 READY
22 link query load(commands(query))  coblib
   READY
23 query
   3288540
   3288540 PAWL SPRING (4-INCH)  13 DOZ ON HAND    7 ORDER POINT
   READY

```

Figure 9. A Terminal Session Creating a COBOL Program

## Fortran

Two versions of FORTRAN IV with special support for the foreground environment are available as Program Products to TSO users:

- Code and Go FORTRAN.
- FORTRAN IV (G1).

Both processors can also be used in the background environment. Two additional Program Products are available to complement the processors:

- FORTRAN IV Library (Mod I), for use with either processor to provide list-directed input/output support, ASCII data set handling, and PAUSE and STOP output to the terminal.
- TSO FORTRAN Prompter, which allows the terminal user to invoke the FORTRAN IV (G1) processor with the FORT or RUN commands.

A FORTRAN programmer can also invoke the FORTRAN (E), (G), or (H) processors with the CALL command, but not with the RUN or FORT commands. The user is responsible for allocating the data sets needed by these compilers, and for specifying the compiler options. The prompter performs these services for the FORTRAN IV (G1) compiler, which also has output specially formatted for the terminal.

Code and Go FORTRAN is optimized for a fast compile-and-execute sequence, carried out entirely within main storage for small-to medium-sized programs. This makes it a useful tool for problem-solvers. It accepts free-form source statements, and has simplified I/O statements for addressing the terminal. However, no permanent object program is produced, and some execution speed is sacrificed for fast compilation. Whenever the programmer need to link separately compiled programs and subroutines, when he is working with very large programs, or when he wants to produce an object program he can save, he will use the FORTRAN (G1) compiler. He may develop and test his program with Code and Go FORTRAN, and then compile it a last time with the FORT command. The TSO CONVERT command will change free form source statements to fixed form or vice versa. Code and Go FORTRAN is discussed in greater detail in the chapter "Problem Solving."

### Entering the Source Program

The programmer uses the EDIT command to create a source program. An operand of the EDIT command informs the syntax checker what FORTRAN compiler is going to be used. As the program source statements are entered, the FORTRAN syntax checker processes each line, interrupting the input sequence if it detects an error. Figure 10 shows a syntax checker diagnostic response and the user action to correct the error. The first CHANGE subcommand inserts a left parenthesis, the second, a right parenthesis.

### Compiling a FORTRAN Program

When the programmer finishes entering the source program, he saves his data set with the SAVE subcommand, and switches to command mode to enter the FORT command, or stays in edit mode and uses the RUN subcommand. Operands of FORT allow him to specify various compiler options: whether or not a listing is to be produced, the contents of the listing, where it is to be printed or stored, whether or not an object program is to be produced, and whether diagnostics are to be sent to the terminal. All operands except the input data set name can default to standard values.

```

.
.
00030    30 format (f10.3)
00040    12 read (2,30) a(i),i=1,5
) REQUIRED FOR IMPLIED DO
EDIT
change / a/ (a/
change /5/5)/
list *
00040    12 read (2,30) (A(I),I=1,5)

INPUT
           do 50 i=1,5
.
.

```

Figure 10. FORTRAN Syntax Checker Diagnostic

As the compiler processes the program, it may find program organization errors that were not detected by the syntax checker on a statement-by-statement basis. Compiler diagnostic messages are sent to the terminal, along with the statement in error, and a pointer to the field in error, if possible. Figure 11 is an example of compiler output to the terminal during a single compilation. The number preceding the source statement is the line number assigned by EDIT when the source program was entered. The line number allows the programmer to use the edit mode subcommands to correct the statement quickly, without listing the entire program.

```

G1 COMPILER ENTERED
000170    30    FORMAT    (I6)
           $
01) IGI006I    DUPLICATE LABEL
SOURCE ANALYZED
PROGRAM NAME=MAIN
*001 DIAGNOSTICS GENERATED,
      HIGHEST SEVERITY CODE IS 8
READY

```

Figure 11. Sample of FORTRAN Compiler Output

When the program compiles successfully, the programmer can print an error-free listing, and use the LOADGO command to load his program for execution.

### Testing FORTRAN Programs

The FORTRAN programmer has two testing facilities: The debug facility of the FORTRAN language, and the TSO test mode.

The debug facility of FORTRAN (G1) allows the programmer to monitor program execution from his terminal. Output from the debug statements such as TRACE and DISPLAY is sent to the terminal, unless directed elsewhere with the UNIT option of the DEBUG statement. DUMP and PDUMP output also goes to the terminal. Execution of the program can be synchronized with the terminal by inserting READ statements in the debug packets, forcing the program to wait for the user to allow it to continue. Since FORTRAN debug statements are grouped together in the source program, they can be easily deleted with EDIT subcommands when testing is completed.

The test mode is also available for FORTRAN programmers. Using an object program listing and storage map produced by the compiler, the programmer can insert breakpoints to interrupt execution of his program, list and modify variable values in main storage, and control program flow. Some knowledge of System/360 instruction formats and hexadecimal notation is helpful in using test mode.

## PL/I

The PL/I programmer can use the following language processors from the terminal:

- ITF: PL/I.
- PL/I Optimizing Compiler.
- PL/I (F) Compiler.
- PL/I Checkout Compiler.

The ITF: PL/I Program Product is a subset of PL/I designed for solving problems at the terminal. It is provided by a compiler that offers two types of processing: a rapid compile-and-execute sequence for small- to medium-sized programs, or line-by-line interpretation and execution of PL/I statements as they are entered. ITF: PL/I does not produce a permanent object program. ITF: PL/I is described in the chapter, "Problem Solving."

The PL/I Optimizing Compiler, an IBM Program Product, is a language processor for use in either the background or the foreground environment. For the foreground environment, the compiler incorporates a prompter, which allows the user to invoke it with the PLI or RUN commands. Compiler options allow the user to request diagnostics and listings formatted for the terminal, or to request termination of compilation if syntax errors are found.

The PL/I programmer can also use the PL/I (F) compiler from the terminal, but no special prompting or output format is available. The F-level syntax checker can be used to scan source statements as they are entered or to scan complete programs. The PL/I (F) compiler cannot be invoked with the PLI or PLIC commands, but an example of a command procedure that uses the CALL command for the PL/I (F) processor is given in the last section of this chapter.

The PL/I Optimizing Compiler implements a more comprehensive subset of PL/I than previous compilers and offers a choice of fast compilation, optimization for speed of object program execution, or optimization for minimum object program size. A subroutine library is required during linkage editing of a compiler output module. A second library is required for execution of the object program. Each library is available as an IBM Program Product:

- OS PL/I Resident Library.
- OS PL/I Transient Library.

The PL/I Checkout Compiler is a two-stage processing program which translates and interprets (executes) PL/I programs. It can be used in either the batch or TSO environment of the IBM System/360 Operating System.

Using the checkout compiler in a TSO environment will often enable you to check out a PL/I program in one session at the terminal. Its conversational checkout features allow you to communicate with the compiler during processing. The compiler prints messages and listings at the terminal (as requested by the TERMINAL option) and you can respond with PL/I subcommands, or PL/I statements for immediate execution. The subcommands allow you to change compiler options, request more information, copy output files at the terminal, make temporary modifications to the PL/I program (during interpretation only), and either continue or terminate processing. The OS/360 PL/I Transient Library is required during execution.

You can also communicate with the PL/I program when it is being interpreted by using the conversational I/O feature of PL/I.

## **Entering a PL/I Program**

The programmer uses the EDIT command to create his source program and save it as a data set. He can request EDIT to assign a line number to each line of his source program as he enters it. If line numbers are assigned, he can request the PL/I Optimizing Compiler or the PL/I Checkout Compiler to use them in diagnostic messages, instead of statement numbers. The programmer can use the line number to retrieve the erroneous source statement, correct the error, and invoke another compilation, all without having the complete listing displayed at the terminal.

## **Compiling a PL/I Program**

To invoke a compiler, the programmer uses either the RUN command, the PLI or the PLIC command. RUN can be used as a subcommand of EDIT, allowing the user to correct errors without entering the EDIT command again. RUN causes a complete compile-load-go sequence but does not produce a permanent object program. RUN is normally used during the initial compilations to check for source language errors. When a program is debugged, the PLI command can be used to produce an object program and a full listing. The object module can be loaded for execution or linkage edited into a program library for use as a load module. Whether invoked by RUN, or PLI, the PL/I Optimizing Compiler directs diagnostic messages to the terminal, in either a full or an abbreviated format. During testing, the programmer can have traces and other output generated by PL/I program checkout facilities displayed at the terminal. The PLIC command invokes the PL/I Checkout Compiler.

## **Program Execution**

Programs produced by the compiler can be executed in either the background or the foreground. In the foreground I/O can be directed to the terminal by allocating a PL/I file, such as SYSIN or SYSPRINT, to the terminal with the ALLOCATE command. In the background these same files can be directed to data sets or unit record devices.

## **Assembler Language**

Like programmers who use the higher level languages, the assembler language user enters his source program statements with the EDIT command. Assembler (F) accepts free form input, but the tab setting facilities of EDIT allow the user to create a formatted listing. On request, EDIT assigns line numbers to the source statements, which are later referred to by diagnostic messages produced during assembly. Line number or context editing is always available to correct errors, modify source statements, or add comments.

## **Assembling the Program**

When the programmer completes his source program and saves it, he invokes Assembler (F) with the RUN or ASM commands. The use of these commands requires the TSO Assembler Prompter Program Product. Operands of ASM give him control over the listing format, disposition of output, and diagnostic messages.

Assembler diagnostic messages sent to the terminal include the statement in error, if possible; both the EDIT-assigned line number and the assembler-assigned statement number; and an explanation of the error. Usually, the user will not need to have the complete listing displayed in order to obtain an error-free assembly. Using the line numbers in the diagnostic messages, the programmer can quickly locate and fix source statements errors with the edit mode subcommands.

## Test Mode

When assembly completes without error, the programmer creates a load module with the LINK command, and uses the TEST command to bring it into storage. The TEST command processor uses the symbol table produced by the assembler and linkage editor, which gives the address and attributes of each symbolic name used in the source program. Before passing control to the program, TEST allows the user to establish initial values to be passed to the program as test data, and to set up breakpoints where execution is to be interrupted for displays, dumps, and other debugging activity. The user can refer to points in the program by symbolic names, absolute relative or indirect addresses.

To display storage and register contents, the programmer uses the LIST subcommand, specifying a register range or address range, or a list of symbolic names. Special forms of the LIST subcommand provide standard formats for control blocks such as the TCB, DEB, DCB, and PSW. LIST will also provide a current map of user storage. List output can be directed to either the terminal or a data set.

Other TEST subcommands allow the programmer to load additional programs into storage, to delete or replace programs in storage, to issue GETMAIN and FREEMAIN as subcommands from the terminal, to define the location and attributes of symbols not in the symbol table, and to start and stop program execution.

## Other Compilers

Any language processor designed to execute under the operating system can be invoked from a TSO terminal. A compiler, like any other program in load module form, is invoked with the CALL command. Options to invoked with the CALL command. Options to control the execution of IBM compilers -- such as LOAD or NOXREF -- are entered with the CALL command, in the same form as they would be specified in the PARM field of an EXEC statement in a background job stream.

Before a language processor is invoked, the necessary input, output, and utility files must be allocated under the names expected by the processor. For the compilers invoked directly by their own commands (American National Standard COBOL, FORTRAN (G1), PL/I Optimizing Compiler PL/I Checkout Compiler and Assembler (F)), the necessary allocations are performed by initialization programs called before the compilers.

Since the ALLOCATE statements necessary for a particular compiler are always the same, it is easiest to define them in a command procedure to be used for invoking that compiler. The function of the command procedure is the same as the cataloged procedures used to invoke compilers in the background: to save the user the trouble of entering a set of unchanging statements each time the compiler is invoked. The command procedure developed in this section as an example is for the PL/I (F) compiler. Similar procedures can be defined, either by individual users or by the installation, for any processor.

## A Compiler Command Procedure

Figure 12 shows a command procedure that could be used to invoke the PL/I (F) compiler. This procedure would be created with the EDIT command as a command list (CLIST) data set, under an appropriate member name, such as PLIF.

```
1  PROC 1 NAME
2  ALLOCATE DATASET( &NAME..PLI ) FILE(SYSIN)
3  ALLOCATE DATASET( &NAME..LIST ) FILE(SYSPRINT) BLOCK(125) SPACE(300,100)
4  ALLOCATE DATASET( &NAME..OBJ ) FILE(SYSLIN) BLOCK(80) SPACE(250,100)
5  ALLOCATE FILE(SYSUT1) BLOCK(1024) SPACE(60,60)
6  ALLOCATE FILE(SYSUT3) BLOCK(80) SPACE(250,100)
7  CALL 'SYS1.LINKLIB(IEMAA)' 'LIST,ATR,XREF,STMT,MACRO'
8  FREE FILE(SYSUT1,SYSUT3,SYSIN,SYSPRINT)
```

Figure 12. A Command Procedure to Invoke the PL/I (F) Compiler

At 1 in the sample procedure is a PROC statement, defining a single positional parameter to be supplied by the user when the procedure is invoked, in this case, the name of his program. Whatever value the user specifies when calling the procedure will be filled into the following commands wherever “&NAME” appears.

Records 2 through 6 perform the data set allocations required by the PL/I compiler. Record 2 allocates the input data set containing the source program. Although this data set is probably already allocated, since the user has most likely just created it with EDIT, this ALLOCATE command will reallocate it with the DDNAME “SYSIN”. This data set is always OLD, no BLOCK or SPACE values have to be supplied. The data set name will be formed from the program name supplied by the EXEC command, followed by the characters “.PLI”. Two periods are necessary in the model command, since the first one indicates the following characters are to be concatenated to the supplied value. Records 3 and 4 similarly allocate and assign standard names to the data sets to hold the program listing and the object program. Since these are new data sets, the BLOCK and SPACE values must be supplied. Records 5 and 6 allocate the two utility, or temporary work, data sets the compiler needs. No data set name is specified, so a system-generated name will be assigned to them, and the data sets will automatically be deleted by a FREE command. All the other data sets will be kept and cataloged. To use the same procedure again for the same program, the user should enter DELETE commands for SYSIN and SYSPRINT.

Record 7 invokes the PL/I (F) compiler by its load module name, and passes to it the list of options to control execution. When the compiler completes processing, the FREE command in record 8 releases all the data sets except the object module.

Figure 13 shows how the procedure might be used from the terminal. At 1 is the EXEC command invoking the procedure. The LIST keyword on the command specifies that each command is to be printed out at the terminal as it is executed. Note that the name supplied with the EXEC command has been filled in as part of the data set name field in the ALLOCATE commands. The system continues to list commands through line 8, then notifies the user it is again ready to accept commands from the terminal with the READY message in line 9. The user enters the LOADGO command to bring his compiled object program into storage for execution.

```

1  exec plif 'exp' list
2  ALLOCATE DATASET(EXP.PLI) FILE(SYSIN)
3  ALLOCATE DATASET(EXP.LIST) FILE(SYSPRINT) BLOCK(125) SPACE(300,100)
4  ALLOCATE DATASET(EXP.OBJ) FILE(SYSLIN) BLOCK(80) SPACE(250,100)
5  ALLOCATE FILE(SYSUT1) BLOCK(1024) SPACE(60,60)
6  ALLOCATE FILE(SYSUT3) BLOCK(80) SPACE(250,100)
7  CALL 'SYS1.LINKLIB(IEMAA)' 'LIST,ATR,XREF,STMT,MACRO'
8  FREE FILE(SYSUT1,SYSUT3,SYSIN,SYSPRINT)
9  READY

10 allocate dataset(*) file(sysin)
11 READY

12 allocate dataset(*) file(sysout)
13 READY

14 loadgo exp.obj pl1lib

```

Figure 13. Use of a Command Procedure

If the procedure is a member of the command procedure library, the user can use the EXEC command implicitly, as shown in Figure 14. When the system does not find “PLIF” defined in the command library, it looks for the command procedure in the command procedure library. The individual commands are not displayed at the terminal. When the procedure completes, the READY message is displayed, and the user can load his program for execution.

```

plif exp
READY

```

Figure 14. Implicit use of Procedure

### Nested Procedures

A command procedure can be made into a compile-load-go sequence -- the equivalent of the RUN command -- by using the procedure nesting and conditional execution capabilities. For instance, in Figure 13, note that the user enters two ALLOCATE commands, defining terminal input and output for execution time, and a LOADGO command to invoke his program. Like the commands used to invoke the compiler, these would normally be used every time the user wants to invoke his program, and therefore can be reasonably placed in a command procedure. This second procedure can be it from the compiler-invoking procedure, making it a compile-load-go procedure.

The procedure to load and execute the user program might be defined as shown in Figure 15, under a suitable name such as LDGO. The FREE command in record 2 is the same as the one in the PLIF procedure. It needs to be repeated here since it will not be executed in that procedure, as explained below. Records 3 and 4 allocate the terminal for an SYSIN or SYSPRINT I/O statements in the user program, and statement 5 is the LOADGO command causing the program to be brought into storage and given control.

```

1  PROC 1,NAM1
2  FREE FILE(SYSUT1,SYSUT3,SYSIN,SYSPRINT)
3  ALLOCATE DATASET(*) FILE(SYSIN)
4  ALLOCATE DATASET(*) FILE(SYSPRINT)
5  LOADGO &NAM1..OBJ PL1LIB
6  END

```

Figure 15. A Command Procedure to Invoke a User Program

It would be possible to call this procedure from the PLIF procedure by inserting a record containing:

```
EXEC LDGO '&NAME'
```

However, it would be preferable to call it only when the return code from the compiler indicates successful execution is likely, that is, no serious errors were detected during compilation. To test the compiler return code, the user inserts a WHEN statement:

```
WHEN SYSRC(LE 4) EXEC LDGO '&NAME'
```

The WHEN statement immediately follows the CALL command invoking the compiler (record 7 in Figure 12). If the compiler return code is less than or equal to four ("LE 4"), minor errors, were detected, the EXEC indicating that no errors, or only command is executed. If the return code is greater than four, the EXEC command will be ignored, the FREE command is executed, and the procedure ends. The terminal returns to command mode, and the user will probably use the LIST command to display the compiler listing, determine the errors in the source program, correct them with the EDIT command, and reinvoke the procedure for another compilation. Figure 16 shows the modified PLIF command procedure. A DELETE command has been added for the object module, since it is not executable. Figure 17 shows a use of the procedure for a successful compilation. The LIST operand is specified to display each command as it is executed.

```
PROC 1,NAME
ALLOCATE DATASET(&NAME..PLI) FILE(SYSIN)
ALLOCATE DATASET(&NAME..LIST) FILE(SYSPRINT) BLOCK(125) SPACE(300,100)
ALLOCATE DATASET(&NAME..OBJ) FILE(SYSLIN) BLOCK(80) SPACE(250,100)
ALLOCATE FILE(SYSUT1) BLOCK(1024) SPACE(60,60)
ALLOCATE FILE(SYSUT3) BLOCK(80) SPACE(250,100)
CALL 'SYS1.LINKLIB(IEMAA)' 'LIST,ATR,XREF,STMT,MACRO'
WHEN SYSRC(LE 4) EXEC LDGO '&NAME.'
FREE FILE(SYSUT1,SYSUT3)
DELETE &NAME..OBJ
END
```

Figure 16. A Command Procedure for a Compile-Load-Go Sequence

```
exec plif 'derv' list
ALLOCATE DATASET(DERV.PLI) FILE(SYSIN)
ALLOCATE DATASET(DERV.LIST) FILE(SYSPRINT) BLOCK(80) SPACE(300,100)
ALLOCATE DATASET(DERV.OBJ) FILE(SYSLIN) BLOCK(80) SPACE(250,100)
ALLOCATE FILE(SYSUT1) BLOCK(1024) SPACE(60,60)
ALLOCATE FILE(SYSUT3) BLOCK(80) SPACE(250,100)
CALL 'SYS1.LINKLIB(IEMAA)' 'LIST,ATR,XREF,STMT,MACRO'
WHEN SYSRC(LE 4) EXEC LDGO 'DERV'
FREE FILE(SYSUT1,SYSUT3,SYSIN,SYSPRINT)
ALLOCATE DATASET(*) FILE(SYSIN)
ALLOCATE DATASET(*) FILE(SYSPRINT)
LOADGO DERV.OBJ.PLILIB
```

Figure 17. Using a Compile-Load-Go Command Procedure

## Problem Solving

To meet the needs of users who may not be professional programmers, three problem-solving languages are available as IBM Program Products with TSO: **Interactive Terminal Facility (ITF): BASIC, ITF: PL/I, and Code and Go FORTRAN**. These languages are available as separate program products. ITF: BASIC is a simple, algebra-like language that can be quickly learned, yet it has the power to perform complex mathematical calculations. ITF: PL/I is a subset of the full PL/I language. It is a more powerful language than BASIC for subroutine handling, but is simpler than the full PL/I language, making it a good teaching tool. ITF: PL/I can be used in two ways: statements can be interpreted and executed as they are entered (desk calculator mode); or they can be collected into procedures for compilation and execution as programs or subroutines. Code and Go FORTRAN provides the full FORTRAN IV language for terminal users. It has a very fast compile-and-execute sequence, carried out entirely in main storage. Code and Go FORTRAN accepts free-form source statements, and has simplified I/O statements for terminals.

All three languages have statement-by-statement syntax checking as the programs are keyed in, and additional diagnostics are set to the terminal for errors detected during compilation and execution phases. For the ITF: BASIC and PL/I languages, the test mode allows the user to monitor program execution with breakpoints and traces, to inspect and reset the values of variables and to modify main storage during execution. The debug facilities of FORTRAN (G) are included in Code and Go FORTRAN.

Programs in any of the three languages are created, and can be run, in edit mode. Whenever necessary, the user can use EDIT to replace or modify source statements. For small to medium-sized programs performance is better in edit mode than in command mode, since the source statements and, in the case of Code and Go FORTRAN, the object program, can be kept in main storage and do not have to be read in from auxiliary storage.

### ITF: BASIC

The ITF: BASIC Program Product is based on the original BASIC language created for time sharing use at Dartmouth College. With TSO, the BASIC user logs on to the system, then enters the EDIT command. In the input mode he enters successive statements to define his problem. If the system detects a syntax error, he is notified immediately so that he can correct the faulty statement before continuing. The user can defer syntax checking until compilation. When all his statements have been entered and syntax checked, the user issued the RUN subcommand to compile and execute the program. An operand of the RUN subcommand specifies whether he wants to execute with short precision (6 significant decimal digits) or long precision (15 digits). Programs and data can be saved from one session to the next, or deleted after use.

BASIC statements are entered one to an input line, and can refer to other statements by the line number assigned by EDIT. Variables always have one- or two-character names. Arithmetic operators used in BASIC statements are +, —, /, \* and \*\* (exponentiation).

BASIC includes statements for defining and handling one- and two-dimensional arrays. Array references have the form A(i,j) where "A" is the array name, and "i" and "j" are variables or constants referring to the row and column of an element. Elements can contain arithmetic or character values.

A special set of statements is included to handle matrices. A BASIC matrix is always a two-dimensional array, and can contain only arithmetic values. Two matrices with the same dimensions can be multiplied, added, or subtracted, and a matrix can be inverted or transposed with built-in matrix functions.

Figure 18 shows a terminal session creating a BASIC program to calculate the infinite sum

$$\sum_{n=1}^{\infty} x^{-n}$$

to the limit of machine precision. Statements 010 and 020 write messages to the terminal, describing the input requested by statement 030. After initializing the variables, the user states the sum as in BASIC format:  $S = S + 1/(X**N)$ . Statement 070 increments N, and 080 is a check to see if the precision limit has been reached. If it has, control branches to statement 100, to print the results at the terminal. Note that statement 110 uses an "image" statement to format output, while statement 130 uses the default format.

During the execution of the program, the "?" requests the user to enter the input. When the results are printed and the program terminates, the user is returned to edit mode where he saves the program for use in later sessions.

```

edit rsumx basic
INPUT
010 print 'summing 1/(x**n)'
020 print 'what x'
030 input x
040 let n=0
050 s=0
060 s=s+1/(x**n)
070 n=n+1
080 if s=s+1/(x**n) then 100
090 goto 60
100 print 'number of terms: |(n-1)
110 print using 120, s
120 : 'sum='##.#####
130 print 'last term=', 1/(x**(n-1))
140 end
150
EDIT
run
      SUMMING 1/(X**N)
      WHAT X
? 1.065
      NUMBER OF TERMS:          176
      SUM=          16.3836700000000
      LAST TERM=          1.53643E-05

save
SAVED
end
READY

```

Figure 18. ITF: BASIC Sample Session

## **ITF: PL/I**

The ITF: PL/I Program Product is a subset of the full PL/I language, suited to problem-solving because of its simplicity and ease of use. For example, there are no arithmetic conversion rules to remember: all data is kept in decimal floating-point format. The language is compatible with PL/I as provided by the PL/I (F) compiler, except that Interactive PL/I does not require semicolons to terminate statements, source language programs are stored with variable-length records, and some arithmetic data formats that would default to fixed-point binary in full PL/I are floating-point decimal in ITF: PL/I. A utility command, CONVERT, is provided to format ITF: PL/I source programs for submission to a batch PL/I compiler, if the user wants to create an object program.

ITF: PL/I can be used under either the EDIT or the CALC commands. Under EDIT, statements are collected into a program. When the program is complete, the RUN subcommand is used to compile and execute it. Under the CALC command, statements are interpreted and executed as they are entered. Statements are discarded as soon as they have been executed. Variables, however, are all defined as “static externals” and kept in a table in main storage, where they can be referred to by subsequent ITF: PL/I statements, or displayed at the terminal. The table of variables created during a session using the CALC command can be saved in a data set for use in later sessions.

Variables included in ITF: PL/I are scalars of either single or double character and bit strings, labels, externals, and entry and return parameters. Execution control statements include DO loops, GOTO branch statements, and IF THEN ELSE conditionals. Procedures collected under the EDIT command can be saved and invoked with the CALL statement, either from another procedure or under the CALC command. Only list-directed and edit-directed stream I/O is provided, either to a file or the terminal. An appropriate set of the PL/I built-in functions is included in ITF: PL/I.

**Test Facility:** When a user invokes an ITF: PL/I or BASIC procedure for execution, as an option he can specify that the program is to be tested. In this case, the system allows the user to set breakpoints in the program before it is started, and to set up program traces and displays of variables. All output from the testing routines is displayed at the terminal. When the program is interrupted by a breakpoint, or when the user hits the attention key, he can display and modify variable values, modify test procedures, and then restart the program at the point of interruption. The ITF testing subcommands are a subset of the TEST subcommands available for the programming languages.

Syntax errors in an ITF: PL/I source statement are detected as soon as the statement is entered, and the user is notified to correct the statement. The user can request deferral of syntax checking to compile time. When operating under EDIT, some errors will only be detected at compile-execute time. In this case, a message is sent to the terminal, and the user is returned to the edit mode to correct the error in the source program.

**Sample Session:** Figure 19 shows a sample terminal session using ITF: PL/I to create a procedure finding the largest common divisor of two positive numbers. “Max” and “min” in statements 50 and 60, and “floor” in statement 110 are built-in functions. Note that since no file is specified in the PUT statements, the output is sent to the terminal. At statement 180, the user entered a null line, indicating a switch from the input mode to edit mode. The SAVE subcommand stores the procedure on auxiliary storage. The user then enters CALC to go to the desk calculator terminal mode, and uses a CALL statement to invoke the procedure.

```

edit div ipli
INPUT
00010  div: procedure(x,y);
00020  /* this procedure is a subroutine that find the */
00030  /* greatest common divisor of any positive x and */
00040  /* y of six digits or less
00050  x1 = max(x,y);
00060  y1 = min(x,y);
00070  if (x1 <= 0)|(y1 <= 0) then do;
00080          put list ('invalid values');
00090          return;
00100          end;
00110  lab: rem = x1 - (floor(x1/y1)*y1);
00120          if rem = 0 then go to out;
00130          x1 = y1;
00140          y1 = rem;
00150          go to lab;
00160  out: put list ('the common divisor is:',y1);
00170  end
00180
EDIT
save
SAVED
end
READY

calc
CALC
call div(9,24)
THE COMMON DIVISOR IS:  3.00000E+00
end
READY

```

Figure 19. ITF: PL/I Sample Session

## Code and Go FORTRAN

For the many problem-solvers who are familiar with the FORTRAN programming language, the Code and Go FORTRAN Program Product is available for use from the terminal. The user creates his program, and optionally has it syntax-checked, with the EDIT command. He uses the RUN subcommand to invoke the Code and Go FORTRAN compiler. The source program is converted to an object program in main storage. As soon as the object program is complete, control is passed to it. The compiler used for Code and Go FORTRAN bypasses certain object code optimization processing for greater compilation speed.

The language includes all the features of FORTRAN IV as defined in the publication **IBM System/360: FORTRAN IV Language, GC28-6515**. Two extensions to the language are included for ease of use from the terminal: free-form source statements and list-directed I/O statements similar to those provided by PL/I.

**Free-Form Statements:** Code and Go FORTRAN does not require statements to begin in column 7. If a statement has a label, the statement can immediately follow the label. If it has no label, it can start in column 1.

A utility command (CONVERT) is available to change free-form source statements to fixed form, if a user wants to submit them to one of the batch FORTRAN compilers after developing and testing them in free form. Code and Go FORTRAN will also accept the conventional fixed format.

**List-Directed I/O:** The list directed I/O facility can be used for any I/O device, but was designed especially for FORTRAN programs that read from and write to a remote terminal. List-directed I/O statements are written with an asterisk (\*) in the field normally used for the FORMAT statement number. Thus, a READ statement to fill an array of five values might be coded:

```
20 READ (5,*) (A(I),I=1,5)
```

When execution reaches statement 20 in the program, the line:

```
00020
```

```
?
```

is sent to the terminal. (This prompting message is suppressed at user request or if the device is not a terminal.) The user can then enter a line such as:

```
30.7 42.85 12.3.29.1 88.43
```

to fill in the array. Input values can be separated by one or more blanks, or by a single comma. Two commas in succession indicate a value is to be skipped. A slash is used to indicate that no more values will be entered, and that any remaining variables in the list are not to be altered. Successive occurrences of the same value can be entered in the form “k\*constant”, as in NAMELIST input. Array A above could have been filled by the line:

```
5*30.7
```

Integer, real, literal, complex, and logical constants can be entered for list-directed READ statements. Real constants can be entered in D, E, or unspecified format. List-directed WRITE statements may not include literal constants, and real constants default to E format on output.

**Sample Session:** Figure 20 shows a portion of a terminal session in which a user writes a program in Code and Go FORTRAN and executes it. With the EDIT command, he specifies he is creating a new program named “VAL”, written in Code and Go FORTRAN (GOFORT), and that he wants it checked for syntax errors as he enters it. READ statement 5 allows the user to enter one of the variables, and to specify how many values will be supplied for the array DELTA. Statement 10 then reads the values for DELTA. The WRITE statement will display the answers from array F at the terminal.

When the program is completely entered, the user shifts from input mode to edit mode by entering a null line. The SAVE subcommand creates a copy of the program on auxiliary storage. The RUN command invokes the CODE and Go FORTRAN compiler, which creates an object program in main storage, calls the loader to resolve external references and bring in necessary library programs, and start the program. Note that the ALOG subprogram is included in the program. The complete FORTRAN subprogram library is available.

When execution reaches statements 5 (00020) and 10 (00030), the user is prompted to supply values, and as the program completes, the answers are transmitted to the terminal. After the program terminates the user returns to the edit, mode, where he could modify the source program if necessary. The saved copy of the program will be kept for the user on auxiliary storage, where it will be available in subsequent sessions.

```

READY
edit val new gofort scan
INPUT
00010  real*4 delta(10)/10*0.0/,f(9)/9*0.0/
00020  5 read(5,*) curnt,n
00030  10 read(5,*) (delta(i),i=1,n)
00040  do 10 i=1,n
00050  10 delta(i)=alog((curnt+delta(i)/curnt)
00060  do 30 i=2,n
00070  30 f(i-1)=delta(i)/delta(1)
00080  n=n-1
00090  write(6,*) (f(i),i=1,n)
00100  end
00110
EDIT
save
SAVED
run
? 00005
176.2 5
? 00010
10. , 17.1, 21.5, 37.4, 127.14
0.167791E 01 0.208565E 01 etc.
end
READY

```

Figure 20. Code and Go FORTRAN Sample Session

## System Summary

This chapter introduces the major control and service routines that have been added to the MVT control program for time sharing. It identifies points where the installation can control system execution, and where modules can be modified or replaced for specialized applications.

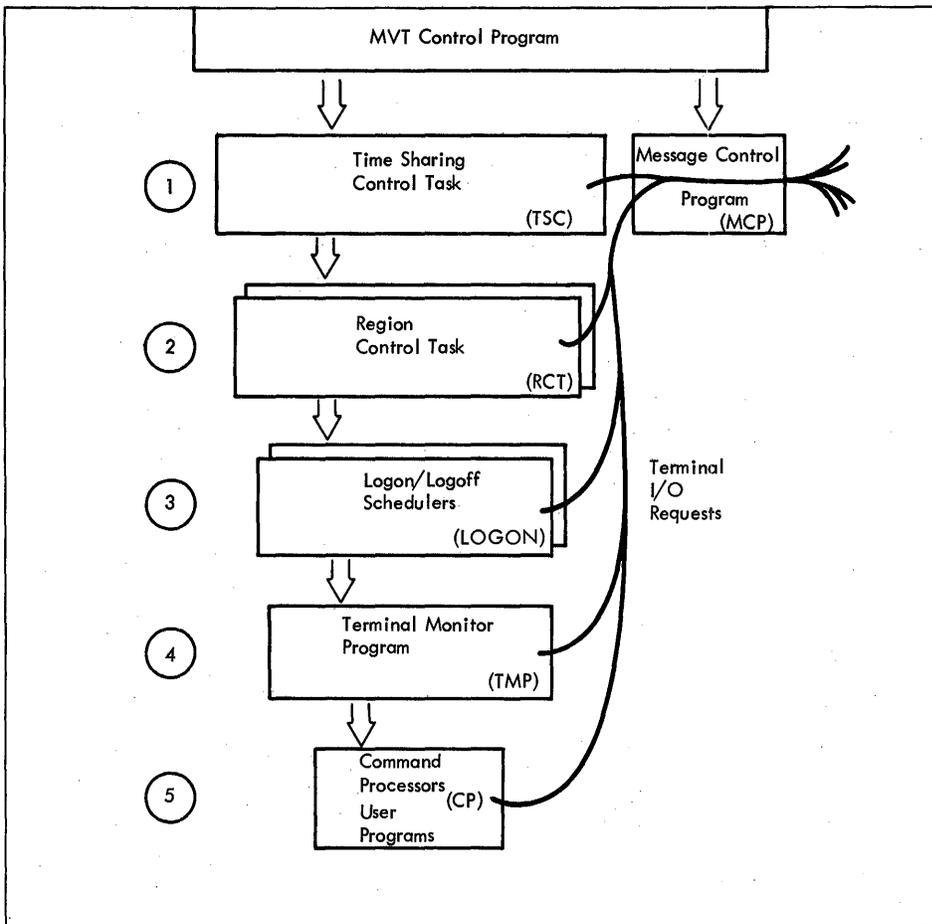


Figure 21. TSO Control Flow Diagram

Figure 21 is a generalized diagram of the flow of control through the system, showing several levels of control under the MVT control program. The portion of the system directly concerned with time sharing can be divided into five levels:

1. At the highest level are the Time Sharing Control and the TCAM Message Control Program tasks. The Time Sharing Control task handles system-wide functions such as the initialization procedure required when the operator starts time sharing, and the swapping of foreground jobs. The Message Control Program is a part of the Telecommunications Access Method (TCAM) and handles all I/O for remote terminals.
2. Below the Time Sharing Control task is the Region Control task for foreground regions. The Region Control task supervises the quiescing and restoring of job activity before and after swapping. Conceptually, there is one Region Control Task for each foreground region, however,

since the Region Control Task is composed of reenterable code residing in the TSC region, only one copy exists.

3. The LOGON/LOGOFF Scheduler is invoked by the Region Control task whenever a user wants to log on or off the system. The LOGON routine identifies the user to the system, and defines his foreground job using parameters in the LOGON procedure, user profile, and operands of the LOGON command.
4. LOGON invokes a problem program specified by the user's LOGON procedure at the next level. This is normally the TSO Terminal Monitor Program, which handles TSO and user-supplied commands.
5. Command processors and other application programs execute at the lowest level of control.

These levels are conceptual only, and are not defined by priorities or locations in main storage. Through the course of this chapter a more precise system flow diagram will be built. However, some overall design features of the system are apparent from even the simplified picture in Figure 21:

- TSO is highly modular -- built up from small components with well-defined interfaces -- and therefore flexible and adaptable to local needs. The Terminal Monitor Program and the Message Control Program are designed to be modified or replaced by the installation for a specialized application.
- Each level of control also provides an opportunity for the system to recover from failure. For instance, abnormal termination of a command processor or other problem program is handled by the Terminal Monitor Program. Only the user who invoked the failing program is affected -- and he is given an opportunity to recover the program through the TEST facility. Users at other terminals are completely protected.

## The Time Sharing Driver

Before discussing the individual control routines in greater detail, one program must be added to the control flow diagram. The Time Sharing Driver isolates in one component the decision-making algorithms for the division of system resources among all the users of the system. By passing parameters to the Driver with the START command or from the system parameter library, the installation controls the various scheduling algorithms to gain the desired performance for its job mix. These "tuning" parameters and the algorithms are discussed in the last section of this chapter.

As shown in Figure 22, the Driver has a unique relationship to the other control routines. It cannot be logically assigned to one of the control levels, but is used as a service program by all the levels from the MVT supervisor down to the Terminal Monitor Program. The calling programs inform the Driver of events throughout the system -- time slice end, user waiting for LOGON, job waiting for input, etc. From this stream of information, the Driver maintains a current picture of the system load and activity. Based on this picture, the Driver orders actions such as swapping, changes in priority, and assignment of a user to a particular region.

The Driver component itself is completely insulated from the rest of the system by the Time Sharing Interface Program, which accepts all calls to the Driver, then passes them through a standard interface to the Driver itself. The Driver returns parameters to the Interface Program that request various actions by the other control routines. Thus, an installation can modify or replace the Driver -- effectively, provide its own system scheduler -- without modifying the system implementation programs. The operator uses the START command to specify which Driver -- the standard one or an installation-written one -- is to be used.

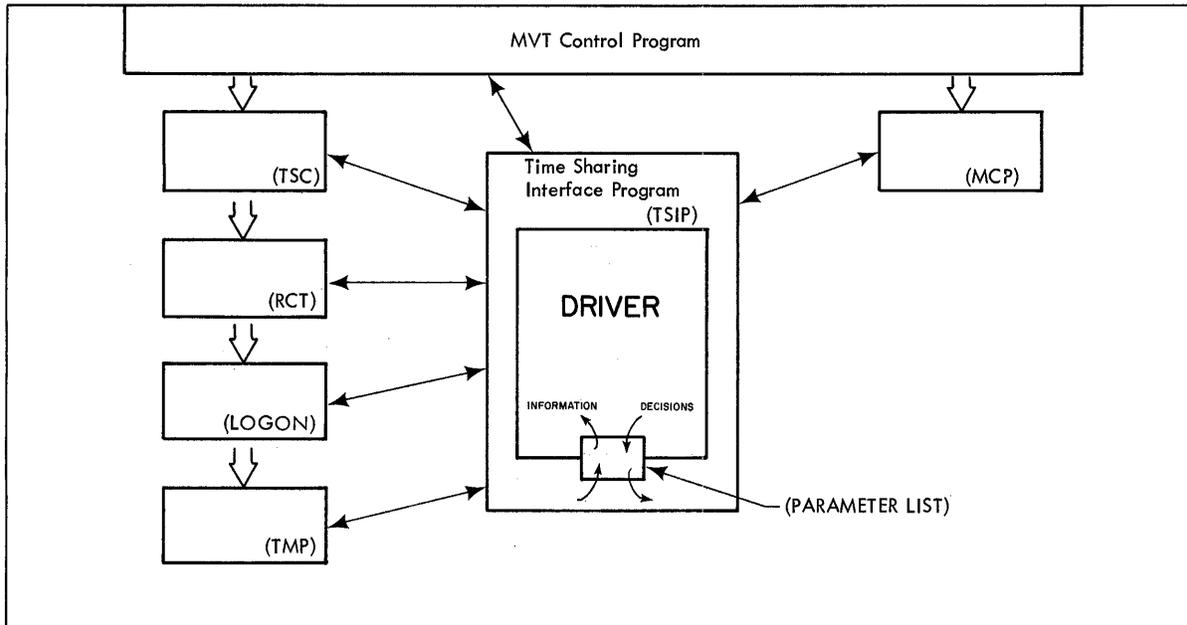


Figure 22. The Time Sharing Driver

## Control Routines

The following paragraphs discuss the functions of each of the TSO routines. Although the TCAM Message Control Program logically shares the highest level of control with the Time Sharing Control Task, it is discussed last.

### The Time Sharing Control Task

The Time Sharing Control task, as shown in Figure 23 handles all functions affecting the entire time sharing portion of the system. This includes responding to the START, MODIFY, and STOP operator commands, and handling the swapping of foreground jobs into and out of main storage.

When the operator enters the START command for TSO, and initialization module of the Time Sharing Control is given control. The initialization module calculates the size of the Time Sharing Control region that will be needed and obtains it from the main storage management routine of MVT. In this region, the Time Sharing Control task builds the control blocks and buffers the system will need, and invokes a Region Control task for each foreground region.

The installation may override the calculated TSC region size by specifying the size it wants in SYS1.PARMLIB or on the START command. This may be necessary if an installation written Driver has greater main storage requirements than the Driver supplied with TSO.

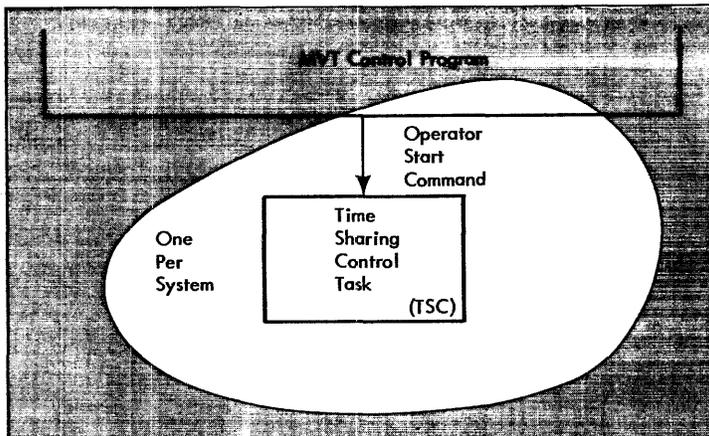


Figure 23. The Time Sharing Control Task

While the time sharing system is operating, the major function of the Time Sharing Control task is the swapping of foreground jobs into and out of main storage. Swapping is handled at this level so it can be optimized on a system-wide basis when multiple foreground regions are active. A swap out is scheduled whether a channel is free or not.

The Time Sharing Control task maintains an input queue and an output queue for swap requests (one of each set if parallel swapping is being used). It builds a channel program for each swap request. A program-controlled interruption (PCI) will occur near the end of each channel program. When the interruption occurs, an exit routine selects the next channel program to execute. The exit routine inserts a transfer to the next channel program at the end of the current channel program. Thus as the number of requests increases, the swap process is carried out by a never-ending channel program. Seek time is minimized by attempting to swap jobs out to the direct access area from which the last job swapped in, or if this is not possible, by using the free space closest to the current arm position.

In determining what portion of a foreground region to swap out, the Time Sharing Control task uses a map of the foreground job created by the Region Control task. Each entry in the map identifies the starting address and length of a section of the region that the job is using. The number of entries in this map is the same for every job and is specified by the installation in the system parameter library. If there are too few entries, inactive main storage must be included (and swapped). A large number of entries cuts down on the amount of inactive storage that has to be swapped, but adds to processing overhead.

When the operator enters a STOP command to shut down the time sharing operation, the Time Sharing Control task initiates a logoff for each active user. When all users are disconnected, the Time Sharing Control task ensures that all the system resources that had been assigned to it are returned; the Time Sharing Control task then terminates, returning its main storage region to the system.

If any users cannot be logged off, the Time Sharing Control task cannot terminate. The operator is given the facility to "force" TSO to terminate even if it appears that normal STOP processing cannot be completed. For further information on "forced" STOP, see message IKJ024D in IBM System/360 Operating System: Messages and Codes, GC28-6631.

## The Region Control Task

A major function of the Region Control task is quiescing and restoring foreground job activity before and after swapping. Conceptually, there is one Region Control task for each active foreground region, invoked by the Time Sharing Control task although only one copy exists in the TSC region. Figure 24 shows a single Region Control task under the Time Sharing Control task.

Before a foreground job can be swapped out of main storage, any activity associated with it must be brought to an orderly halt, or set up to be handled by some supervisor routine that will be remaining in main storage. This includes removing control blocks associated with the job from system queues, or flagging them as inactive.

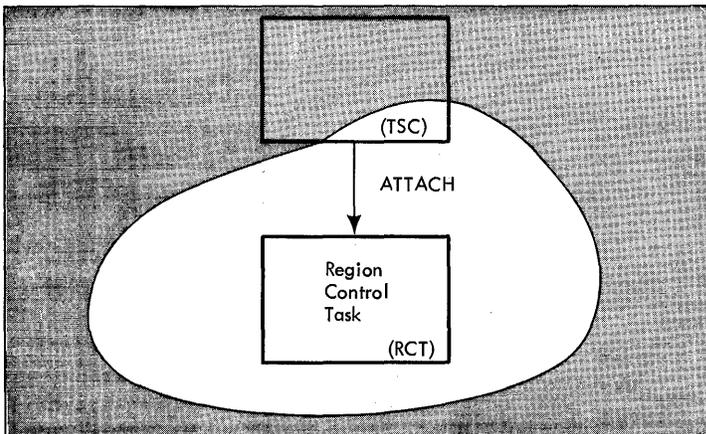


Figure 24. The Region Control Task

Quiescing of I/O activity is initiated by the Region Control task (at the request of the Driver), which issues the Purge Supervisor Call for each task associated with the foreground job. The Purge routine removes I/O requests from the I/O Supervisor's queues of pending requests if they have not yet been initiated. If a request has been started, that is, if data transfer is already taking place, it is allowed to complete before the job is marked ready for swapping. The control blocks associated with unstarted requests are stored in the foreground region where they will be swapped out of main storage along with the job.

I/O requests that address the terminal are an exception to the quiescing procedure because of their long completion time. These requests are handled through the TSO interface with TCAM and are buffered in supervisor main storage, not in the foreground region. Data can be written or read to these buffers whether the job is present in its main storage region or not.

Many control blocks, like the I/O requests mentioned above, reside in the foreground region. For background jobs, these control blocks would be created and maintained in the System Queue Area, a section of main storage set aside for this purpose during nucleus initialization. Foreground regions, however, each contain a Local System Queue Area to hold control blocks. As part of quiescing, the Region Control task removes pointers to these control blocks from system queues. The blocks can then be swapped out of main storage along with the foreground job. The only control blocks for foreground jobs that are assigned in the System Queue Area (and remain in main storage) are requests for timer interruptions, operator replies, and assignment of resources through ENQ.

When a job is swapped into main storage by the Time Sharing Control task, the Region Control task receives control to restore the I/O requests it intercepted at swap out time, and to return the control blocks associated with the job to the appropriate system queues.

## LOGON/LOGOFF

The LOGON/LOGOFF Scheduler routine performs the same functions for foreground jobs that the reader/interpreter and initiator do for background jobs. When defining a foreground job, LOGON uses many of the same programs as subroutines.

When LOGON is invoked by the Region Control task, as shown in Figure 25, it is swapped into the foreground region. A copy of the LOGON/LOGOFF Scheduler for each foreground region is kept in the swap data set, reducing the amount of initialization time needed. LOGON, and all routines below it in the control flow diagram, execute from the foreground region, and are swapped in and out of main storage. LOGON first validates the user's identification and password in the User Attribute Data Set, and reads in the rest of the user profile. From the profile and any operands entered with the LOGON command, LOGON builds, in main storage a JOB and an EXEC statement that define the foreground job. The EXEC statement names the LOGON Procedure specified by the user, and the procedure in turn specifies the name of the program to be invoked. The procedure also contains DD statements for data sets the user always wants allocated to him, and some special DD statements that save control block space for data sets he may allocate later, dynamically.

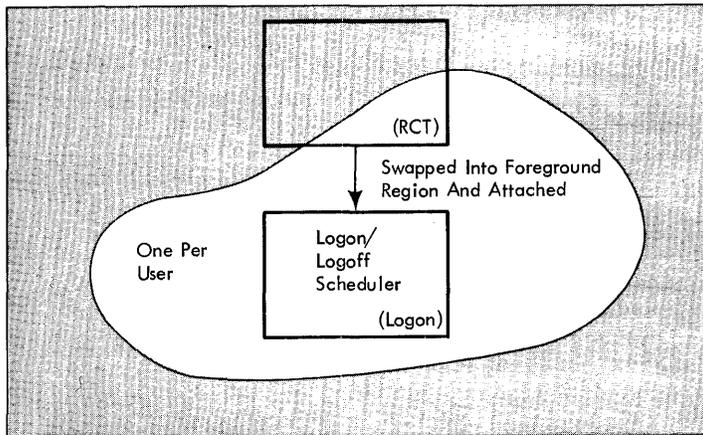


Figure 25. The LOGON/LOGOFF Scheduler

The JOB and EXEC statements built by the LOGON routine are passed to the reader/interpreter to define the resources required by the job, and then to the initiator for allocation of the resources -- direct access storage space, main storage control blocks, etc., -- and invocation of the program. Figure 26 shows the linkage scheme used during LOGON. Use of the system reader and initiator ensures that foreground jobs are compatible with normal background jobs, appearing to the system as a job consisting of a single step.

The LOGON/LOGOFF Scheduler executes in the user's foreground region. Assignment to that region is provisional until LOGON determines the correct size for the user's needs. If a larger region is appropriate, LOGON can, through the Region Control task and the Time Sharing Control task, request that the Driver assign a different region. If a region switch is made, the job information LOGON has already gathered is left in a supervisor buffer. The Driver, through the Time Sharing Control task, causes LOGON to be invoked in the new region, where processing can proceed.

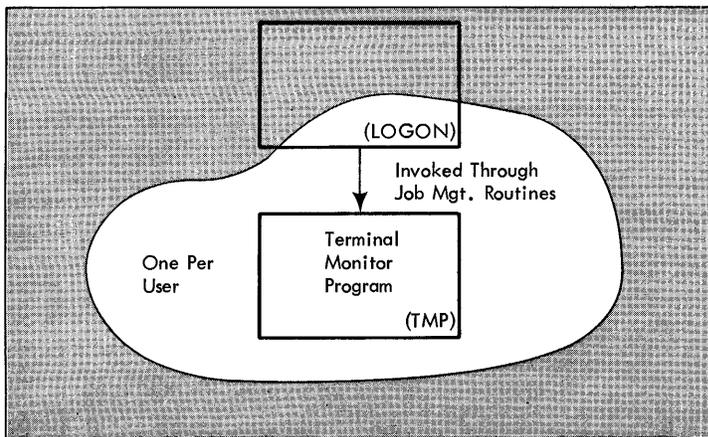


Figure 26. LOGON Linkage

The LOGON routine is also brought into a region whenever a user enters a LOGOFF command, or a second LOGON command, during a session. For logoff, the LOGON routine ensures that all resources have been returned to the system, and calls the accounting routines to update the user's statistics. A second LOGON command during a session also causes logoff processing, but it is immediately followed by re-logon. The new LOGON may request a different LOGON procedure or region size.

### The Terminal Monitor Program

For users of the TSO command language, the program named in the EXEC statement of the LOGON Procedure is the Terminal Monitor Program. Users of locally-provided command systems, or terminal monitors "dedicated" to some local application, can specify these programs in the LOGON Procedure. If necessary for security reasons, user access to particular applications can be controlled through the profiles in the User Attributes Data Set.

The remainder of this discussion concerns only the IBM-supplied Terminal Monitor Program, as shown in Figure 27. Installation-written monitors must perform similar functions, and can use some or all of the service routines described below.

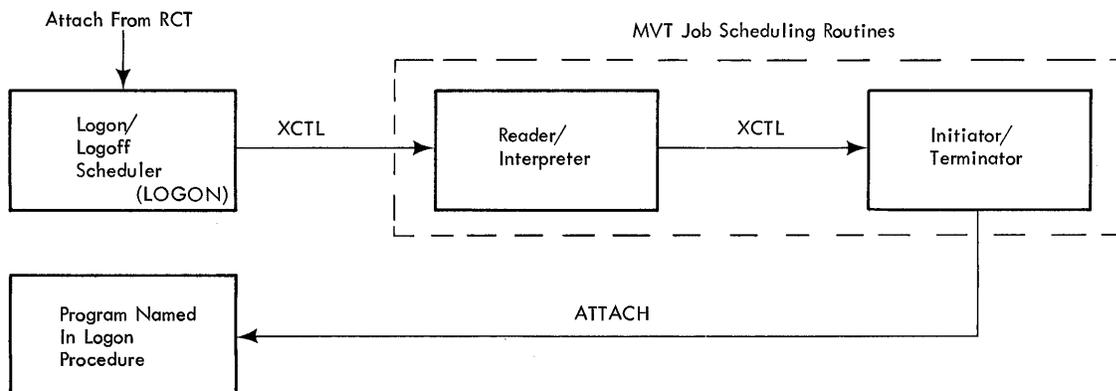


Figure 27. Terminal Monitor Program

When the Terminal Monitor Program receives control from LOGON, it is passed a pointer to the user profile. The profile contains information to control the environment of the current session -- the user identification to append to data set names, whether the user wants to be prompted for command information, whether he wants numerical message identifiers included in messages to the terminal.

During a session, the Terminal Monitor is called on to handle four conditions:

- A command processor or user program is completing, and a new command must be requested.
- A command processor or user program is terminating because of an error.
- The user hit the Attention key, interrupting the current program.
- A CANCEL operator command is forcing a LOGOFF for the user.

To invoke a command processor, the Terminal Monitor Program uses the command name to search the command library or libraries for the processor load module. When it is found, an ATTACH macro instruction is used to invoke it. When the command processor completes, the Terminal Monitor issues a DETACH for it, and writes a READY message to the terminal, indicating another command may be entered.

When a command processor or a user program invoked by a command (CALL, RUN, etc.) terminates because of an error, control is passed to a Terminal Monitor Program routine that notifies the user of the error condition and allows him to enter a new command. If the new command is TEST, abnormal termination processing (ABEND) is cancelled and control is passed to the TEST processor so the user can examine the failing program and attempt to recover. If the new command is not TEST, the failing program completes abnormal termination and the new command is processed.

When a user hits the attention key, or when a attention interruption is simulated for terminals without an attention key, the Terminal Monitor Program attention routine is given control, unless the currently executing program (a command processor or user program) has specified an attention-handling routine of its own. The Terminal Monitor Program attention routine gets a line from the terminal. If it is a program status inquiry such as TIME, the Terminal Monitor Program handles the inquiry and does not cancel the interrupted program. If a new command is entered, the interrupted program is cancelled and the Terminal Monitor Program invokes the new command processor. If the user enters a null line, the interrupted program is restarted at the point of interruption although the current content of the buffers are lost.

When the operator or a control user enters the CANCEL command to force a user logoff, the Terminal Monitor Program terminates any program the user may have running, and returns to the LOGON routine for logoff processing and accounting.

## TEST

The TEST processor is handled differently than other command processors, since it must be able to control the execution of programs (including command processors) being tested. The TEST routine executes at the same level as the Terminal Monitor Program -- receiving control via a LINK, rather than a ATTACH, macro instruction.

TEST reads successive subcommands from the terminal or a command procedure. Each subcommand requests some action -- modification of the tested program's registers or storage areas, insertion of breakpoints in the program, display of data. When the GO subcommand is encountered, the tested program is allowed to execute to the next breakpoint (an inserted SVC instruction) or to completion. When a breakpoint is encountered, TEST again receives control. It will then handle subcommands specified previously by the user, new subcommands entered from the terminal, or both. Another GO subcommand will restart the tested program.

## Service Routines

The command processor service routines, as shown in Figure 28, are used by the Terminal Monitor Program, TEST, and command processor. In general, they perform services that are useful to all foreground programs, and their availability as subroutines saves repetitive coding in all the command processors. They can be called from programs written in Assembler language.

**Command Analysis:** Two routines are provided for analyzing input lines to the Terminal Monitor Program and command processors. The Scan routine determines if an input line contains a syntactically correct command name, and, if it does, returns it to the calling program. The Parse routine continues the analysis of a command or subcommand by comparing the line to a parameter supplied by the calling program describing the permissible operands and default values. The Parse routine builds a new parameter list from this information, describing the options the user has selected, and returns it to the calling program.

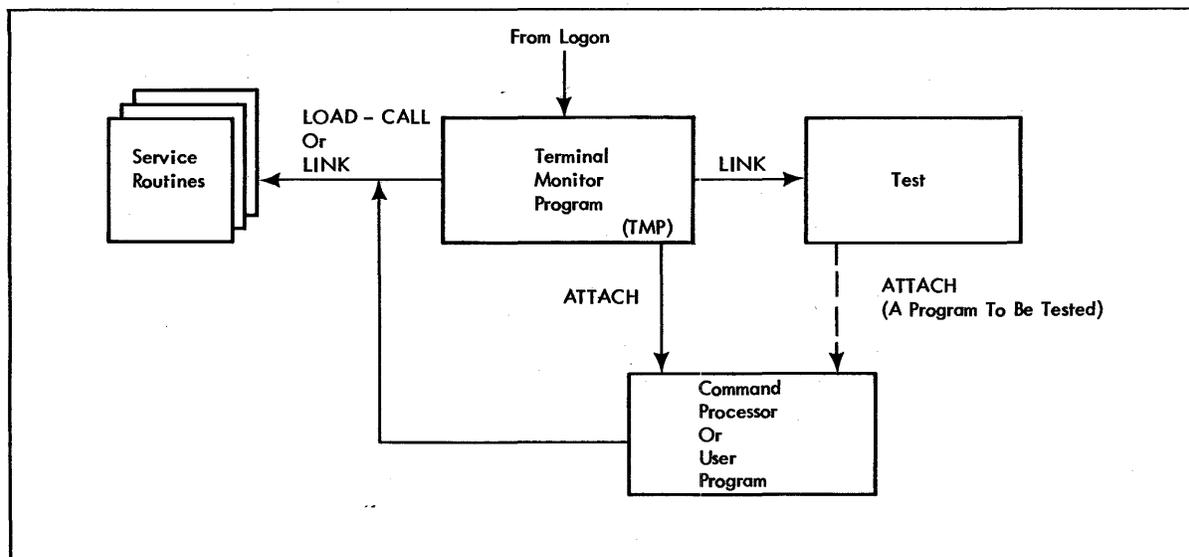


Figure 28. Service and TEST Routine

**Terminal I/O:** Four service routines are provided to handle command processor input and output for the terminal. Command processors normally accept input lines containing subcommands and data from the terminal, and send messages back to the terminal. However, a command processor may be invoked from a command procedure, and in this case, the input to the command comes from an in-storage list built by one EXEC command processor from the CLIST data set that contains the procedure. To allow the command processors to be independent of the source of input, I/O is handled through the Getline, Putline, and Putget service routines.

Getline, Putline, and Putget use a push-down list to keep track of the current input source. Entries in the list represent a terminal, or an in-storage list. The in-storage list may be a command procedure or data. A fourth service routine, Stack, is provided to manipulate this list as the input source changes.

When a command processor calls Getline for a line of input, Getline checks the list of sources to determine the current source and returns one record from that source. The caller need not know whether the input came from the terminal or an in-storage buffer. Putline also checks the list of input sources before sending output from the command processor. Some types of messages, identified by a code in the message identifier, are suppressed if the current input source is not the

terminal. For instance, it is not appropriate to issue a prompting message for command operand information if a command procedure is in progress. A return code to the caller indicates whether the message was issued or suppressed. Putget combines the function of Putline and Getline -- first sending a message, then returning one record.

**Dynamic Allocation:** The Dynamic Allocation Interface routine handles data set allocation and manipulation for command processors. Dynamic allocation uses control block space reserved by DD DYNAM statements in the user's LOGON Procedure. These control blocks are used over and over again when different data sets are needed, but at any one time, a user can have only as many data sets allocated as he has DD statements in his LOGON Procedure. Command processors call on Dynamic Allocation to allocate data sets, to free data sets, to search the system catalog for a particular data set or group of related data sets, and to concatenate or separate groups of data sets.

In TSO a problem may arise from the multiple use of a Job File Control Block (JFCB) for an input data set. When the data set is opened, information supplied in the Data Set Control Block (DSCB) and the JFCB is used to fill any zeroed fields in the Data Control Block (DCB). The opening routines then do a reverse merge from the DCB back into the JFCB, this time filling any zeroed fields in the JFCB. If the same data set is subsequently opened using another DCB, the opening routines will retrieve information from the JFCB for fields not specified in the DSCB or on the DD card. This information could be faulty and could cause a program failure. Deleting the data set with the DELETE command and allocating it again with an ALLOCATE command will prevent these kind of errors.

#### **Command Processors and User Programs**

Although command processors vary widely in function, they have some initialization features in common. All call on the Parse routine to analyze the invoking command and prompt the user for missing or invalid operands, and all use the Dynamic Allocation routine to determine if necessary data sets are allocated and to allocate them if they are not.

At this point, some command processors call on standard system processors to carry out the function desired. For instance, the TSO COBOL Prompter sets up a standard calling sequence according to the options selected by the user, and transfers control to the American National Standard COBOL compiler to compile the user's program. Except for the special formatting of output and messages, the compiler operates exactly as it would in the background.

User-written command processors, and other programs that are not defined as command processors, should avoid using the special Terminal Monitor Program-command processor interface if they are to be compatible with the background environment. The CALL, LOADGO, and RUN commands allow control information to be passed to background-compatible programs in exactly the same format as information in the PARM field of an EXEC statement. Data set allocation can be handled by ALLOCATE commands in a command procedure used to invoke the program.

## Terminal I/O

The Telecommunications Access Method, or TCAM, handles all I/O between remote terminals and jobs in the system. TCAM distinguishes between time sharing applications, with emphasis on direct control of the calling terminal, and other teleprocessing applications, where emphasis may be on queuing, formatting and routing of messages between remote terminals or between applications and remote terminals.

### The Message Control Program

The Message Control Program is the TCAM control routine. It contains definitions and descriptions of the various terminals that can connect to the system, it has buffers for storing data going to and coming from the terminals. It transfers data between its buffers and time sharing buffers.

Most of the Message Control Program is written using a special set of macro instructions that is essentially a language suitable for defining the telecommunications network and specifying the handling of messages on the network and in the system. Macro instructions to generate a Message Control Program suitable for handling terminal I/O for time sharing are distributed with the TSO package.

The Message Control Program executes as a problem program, in a main storage region with a nonzero protection key. Normally, it has the highest priority of the problem programs in the system. It must have a higher priority than the Time Sharing Control Task.

### Mixed Environment MCPs

A TSO message control program can contain more than one message handler. A message handler is a sequence of code that routes terminal I/O to the appropriate program or terminal. A mixed environment Message Control Program contains the message handler for terminal I/O for TSO and in addition one or more non-TSO message handlers.

In a mixed environment, the terminals used with TSO are allocated to the TSO message handler and the terminals used for TCAM applications to the TCAM message handlers. This is done through the macro instructions which define the message control program and through the catalogued procedure that starts the message control program.

### Terminal Interfaces

A variety of interfaces to Terminal services are provided in TSO. The one suitable for a particular program depends on whether the program is defined as a command processor, and whether it must also be able to execute in the background environment.

A supervisor call routine, reached through the TGET and TPUT macro instructions, provides a direct route for program I/O to a remote terminal. TGET and TPUT transfer data between the calling program and a set of buffers in the Time Sharing Control Task region, that are, in turn, emptied and filled by TCAM. Through TGET and TPUT, the calling program can control deletion or insertion of terminal control characters, and whether an output transmission is to break in on any input transmission in progress. A program using TGET and TPUT does not have to perform OPEN or CLOSE processing, and need not provide a DCB for the terminal. However, these macro instructions are available only to programs executing in the foreground.

Programs designed to be command processors can call on the Getline, Putline, and Putget service routines used by the IBM-supplied command processors for I/O. As noted earlier, these service routines have the capability to switch the input source from the terminal to a buffer in main storage, and to suppress certain types of output if the terminal is not the current input source.

The sequential access methods, BSAM (READ, WRITE, CHECK) and QSAM (GET, PUT), have been extended to call on TCAM (TGET, TPUT) when called from foreground programs for terminal I/O. This is the normal route for terminal I/O from programs that must be executable in the background as well as the foreground, or which are coded in a higher level language, such as FORTRAN or COBOL.

Programs using BSAM and QSAM to reach the terminal use the standard macro instructions or I/O statements. When the program is executed, the DD statement or ALLOCATE command defines whether the I/O is for a data set or the terminal. No recompilation is necessary to switch from one to the other, only a change in the DD statement.

Getline, Putline, Putget and the sequential access methods all issue TGET or TPUT for the caller when the I/O is for a terminal. Figure 29 shows this SVC routine handling calls from anywhere within the TSO system and passing the requests to the TCAM Message Control Program.

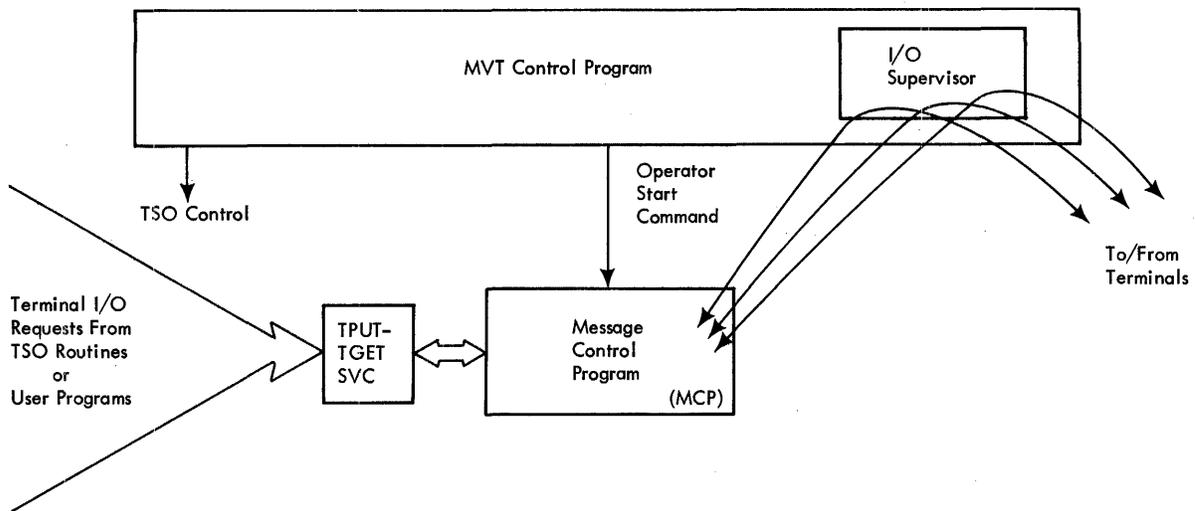


Figure 29. TCAM Message Control Program

### Multi-Terminal Message Processors

Independent of TSO, the Telecommunications Access Method includes facilities for routing messages received from remote terminal to queues for an application program, and transmitting replies generated by the applications program to queues for a terminal. In a system without TSO, such a message processing program must reside in main storage in one of the problem program regions, when it is to be available if one of the terminals in the telecommunication network sends a message that requires processing. With the addition of TSO, a terminal user logged on to TSO can execute a TCAM message processing program in a foreground region. He can do this by invoking it through the TSO command language, or by specifying it instead of the TSO Terminal Monitor Program on his LOGON procedure. The DD statements which define the process queues must be contained in the LOGON procedure. The program will be swapped in whenever needed, but will not occupy main storage space when it has no message to process. Unlike standard foreground jobs, which are associated with a single terminal, these message processing programs can handle GET/READ, PUT/WRITE TCAM oriented input/output from any terminal defined to the TCAM processing queues, through the QNAMES operand of the statements on the LOGON procedure. In addition, the standard TSO terminal interfaces, can be used to interact with the terminal executing the Message Processing Program. For further information on message processing programs, see **IBM System/360, TCAM Programmer's Guide and Reference Manual, GC28-2024.**

## Overview and Storage Map

Figure 30 is an overview of the complete time-sharing system as developed in the preceding sections. The picture is simplified in that it shows only one task at each level of control; there is actually one LOGON-Terminal Monitor Program for each user. Many of the programs themselves are re-enterable, and can be placed in an extension to the link pack area (LPA) built when the operator starts TSO.

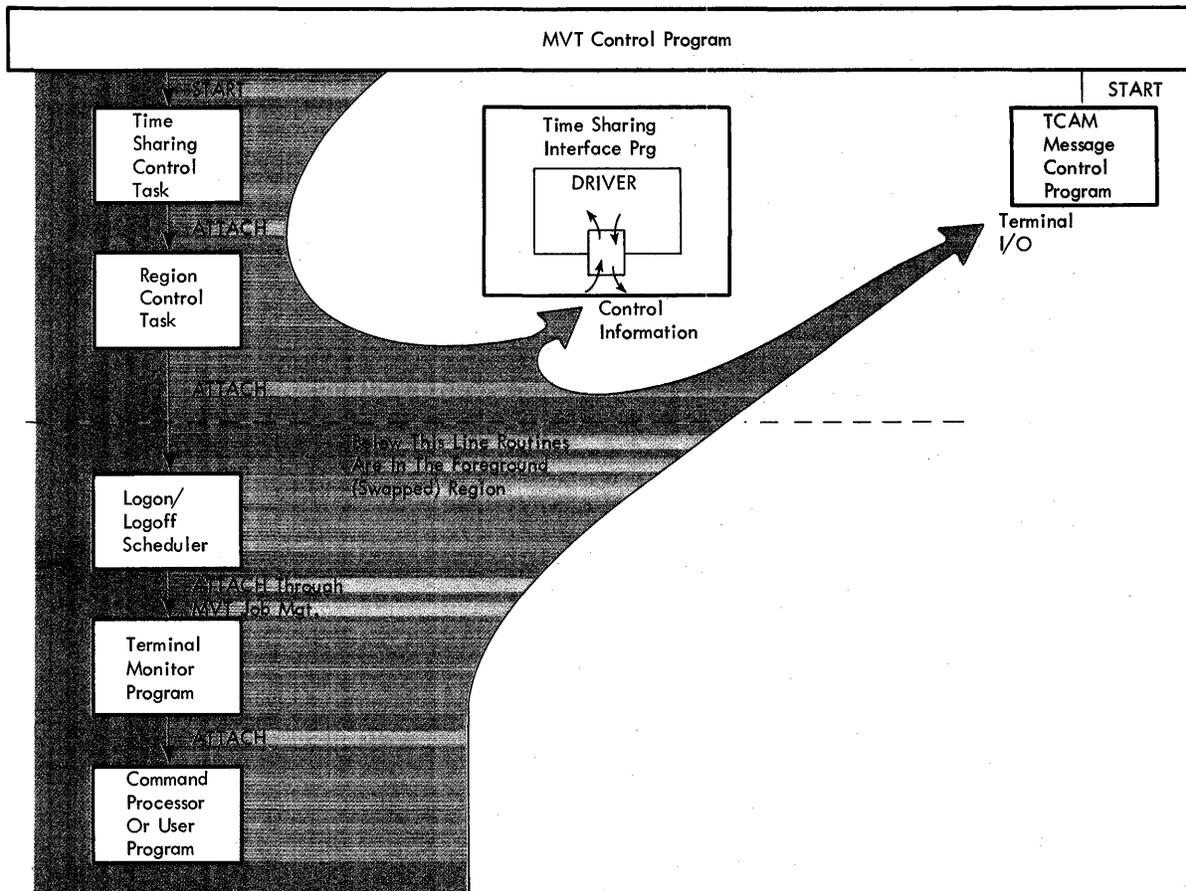


Figure 30. System Overview

Figure 31 is a map of a typical main storage layout when TSO is operating. Almost all the additional storage requirement for TSO control functions is included in the Time Sharing Control Task region. This region is not assigned until the operator enters the START TS command, so the presence of TSO in the system has no effect on MVT throughput when time sharing is not active.

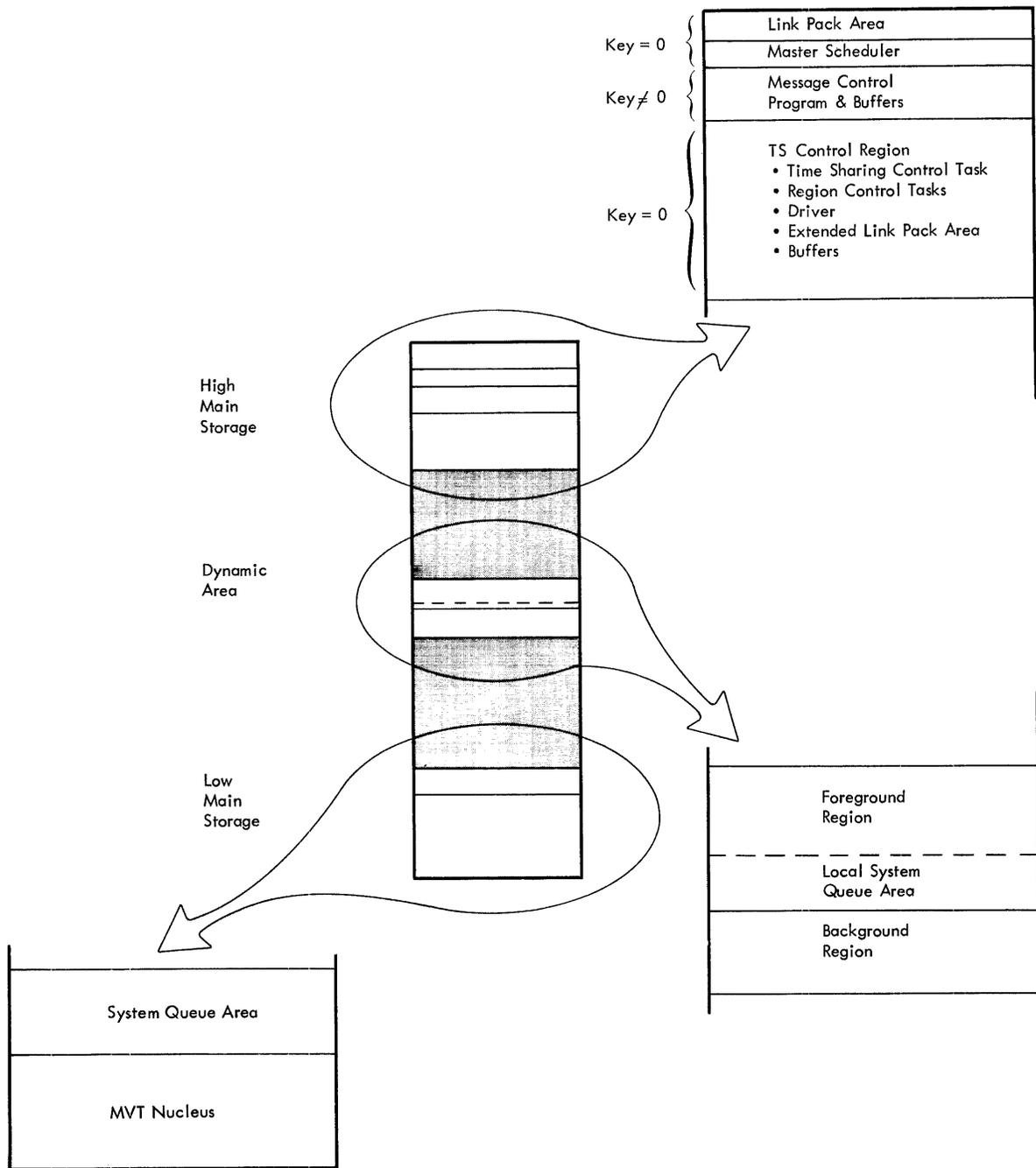


Figure 31. Typical Main Storage Map

## Time Sharing Algorithms

As noted earlier in this chapter, the Time Sharing Driver is responsible for dividing the system resources -- most importantly, execution time -- among the various jobs in the system. So that this may be done effectively, the Driver is given a constant stream of information about the status of each job in the system -- whether it is ready to execute, whether it is waiting for I/O, whether it is in main storage or has been swapped out.

In a time sharing system, execution time is divided among the active foreground jobs and background jobs in brief time slices. A time slice must be long enough to perform a meaningful amount of processing, but not so long that the time between successive slices prevents quick response to conversational users. At the same time, time slices cannot be so short and frequent that system overhead for swapping and task switching becomes unreasonable.

Balancing these factors depends partly on the number and type of jobs the system is processing: a solution for one job mix is not necessarily suitable for another job mix. The Driver's time sharing algorithms -- the formulas it uses to calculate the division of execution time among the jobs in the system -- are based on several variables, many of which can be specified by the installation to tune the system for the local job mix. These variables may specify the system configuration, such as the number of foreground regions to be activated; they may request the Driver to use one of several algorithms it has for a particular calculation; or they may specify constants used in the algorithms. The variables are stored in a member of the system parameter library.

### Time Slices

The Driver uses two important cycles in calculating time slices. One is the cycle of foreground jobs assigned to a region being swapped into the region, then back out to the swap data set on auxiliary storage. The average length of time to complete one cycle -- swapping each job assigned to the region into it one time -- can be controlled by the installation for each foreground region. The length of time each job gets to remain in main storage during a cycle is called the major time slice. Figure 32 shows a cycle of major time slices and swapping of jobs between the main storage region and a swap data set.

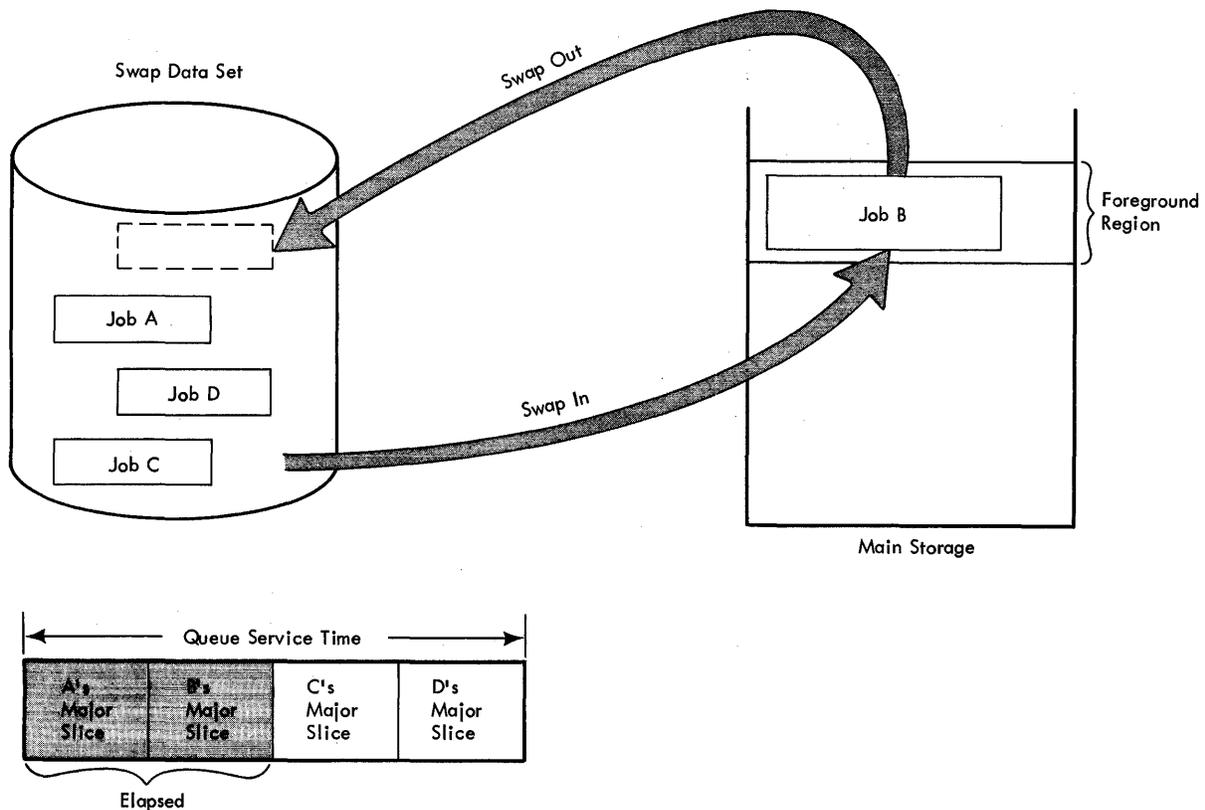


Figure 32. Queue Service Time

The other cycle used by the Driver is the allocation of execution time to the jobs in main storage. At a particular time there are likely to be several regions containing jobs ready to execute -- one or more foreground regions containing jobs swapped in for major time slices, and some background jobs in their own regions. The Driver divides the amount of time remaining until the next scheduled swap out among the jobs that are ready to execute, resulting in a minor time slice for each. For the duration of its minor time slice, each job has the highest effective priority of the problem programs (excluding TCAM). As in batch MVT, if the job cannot execute because it is waiting for I/O or some system resources, another job runs until the higher-priority job is ready again.

Figure 33 shows the indirect relationship between major and minor time slices. A major slice is a fraction of a cycle of swaps into a foreground region, and is the length of a job's stay in main storage. The minor slice is a fraction of the time remaining before the next scheduled swap out for any region (called the available execution time), and determines how long each job will remain at the highest effective priority; that is, how much execution time the job is allotted.

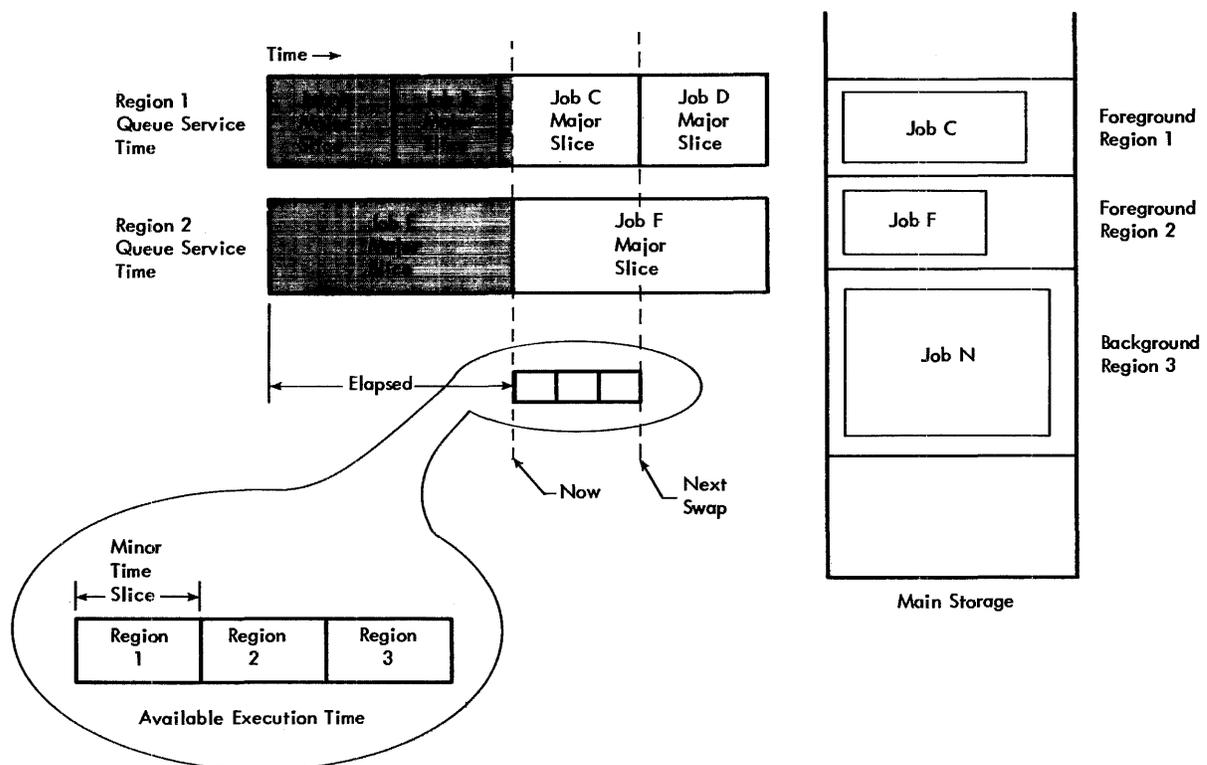


Figure 33. Minor Time Slice

Major and Minor time slices can be calculated using only the number of ready jobs, and the available execution time. However the Driver algorithms have the capability to distinguish among varying user needs to provide the best service to each foreground job. The following two sections show how the tuning variables can be used to make the calculation of time slices most efficient for varying job mixes.

### Major Time Slices

Swapping all the jobs into a foreground region from a queue of ready jobs assigned to that region is called servicing the job. The length of time used to swap all ready jobs on one queue is called the queue service time. The average queue service time for each queue of foreground jobs is an

installation parameter, passed to the Driver. The specified queue service time is divided by the number of ready foreground jobs on the queue, yielding a major time slice value for each job for that service cycle. As the number of jobs assigned to that queue increases, the major time slice value gets smaller. The time between services for each job remains fairly constant, which is important for conversational users expecting quick responses.

A problem may arise when a large number of users are assigned to the queue. The division of queue service time may result in a major time slice too short to perform any meaningful amount of processing for the user, and the system will be spending all its time swapping. To avoid this condition, the installation specifies a minimum major time slice for each queue. Each job is guaranteed at least that amount of time in main storage on each cycle (provided it is ready to execute). When the minimum slice is being used, the actual queue service time will exceed the specified average queue service time.

**Multiple Region Queues:** To meet varying needs of users performing different kinds of processing, the installation can establish multiple service queues for each foreground region. Queue service cycles can be rotated equally among these queues, or priorities can be specified among them. Each queue is assigned its own average queue service time. The installation can also specify that a queue is to be given multiple cycles before the next queue is serviced, or that it is to be serviced until empty -- that is, until no jobs are left on the queue that are ready to run.

Assignment to queues can be based on the amount of main storage the job is using, or the degree of interaction with the terminal, or both. The amount of main storage assigned to the job is called **swap load**, since it is a measure of the amount of I/O necessary to swap the job in and out of storage. A swap load limit can be specified for each queue. If a job's storage needs grow beyond the limit, it is assigned to a lower queue, with a higher limit. The lower queues can be set up to receive fewer services, but longer major time slices at each service. Therefore the larger jobs will not have to be swapped so often.

The degree of interaction with the terminal is measured by the amount of processing time used by the job since its last request for I/O to the terminal. A terminal I/O request is called an **interaction**, and the length of execution time between interactions is called **interaction time** or **occupancy**. Very long interaction times indicate the user is not currently processing conversationally -- perhaps he is compiling a program, or executing some long-running problem program. In this case, his job does not require the quick response times provided by the higher region queues, and can be moved to a lower queue where it will receive fewer, but longer, major time slices. Each queue can be assigned an **interactive time limit**, to allow for this differentiation. The occupancy and swap load limits for a given queue must be higher than the next lowest queue.

Either the swap load or the interaction time limits can be suppressed when the time sharing operation is started for the day, but if both are suppressed, only one queue per foreground region is maintained. The rotation of service cycles around the queues for a region can also be made preemptive: any time a job on higher queue becomes ready to execute; for instance, if a terminal I/O request completes, the service cycle of any lower queue is interrupted to service the job on the higher queue. This scheduling scheme tends to make responses to trivial terminal requests very fast, while lengthening somewhat the response to requests requiring a lot of processing time.

**Region Assignment:** The last factor involved in calculating major time slices is choosing a foreground region for the user logging on. The minimum region size needed by the user is stated in this LOGON procedure, and only those foreground regions large enough are considered.

If a choice must be made among two or more regions, the system can try to balance the workload by assigning the new user to the region with the fewest logged-on users. However, this leaves open the possibility that a group of users all requiring a lot of execution time will be assigned to one region, while another region has a preponderance of users processing conversationally.

Neither group will receive the best service possible. To prevent this condition, the installation can specify that an **average region activity** be maintained for each foreground region. The average region activity is the number of jobs likely to be ready to execute (not waiting for terminal I/O, for instance) at the beginning of the next cycle of major time slices. A new user is then assigned to the region with the lowest region activity, which is not necessarily the region with the fewest logged-on users.

The region activity estimate is based on the number of ready jobs on the region's queues during recent major cycles. Values from more recent cycles are "weighted" in calculating the average. The weighting factor, called the "region activity decay constant," is specified by the installation. Use of decay constant prevents wild fluctuations in the region activity because of a few cycles, but allows gradual change to reflect changing workload.

**Major Slice Variables:** To summarize, the system programmer can specify the following variables affecting the major time slice calculation:

- The maximum number of users logged on.
- The number of foreground regions.
- The method for assigning users to regions.
- The number of service queues for each region.

For each region queue, the following variables can be specified:

- The average queue service time.
- The number of service cycles before advancing to the next queue.
- The minimum major time slice.
- The swap load limit.
- The interaction time limit.

Variables can be omitted or ignored, if the job mix and workload allow simplification of the algorithm. In general, the more homogeneous the job mix, the more the algorithm can be simplified, dividing time almost equally among the jobs. Remember that the major time slice determines only how long a job remains in main storage, not how much execution time it receives. Calculation of the minor time slice, which determines execution time, is discussed next.

### **Minor Time Slices**

The minor time slice is the result of dividing the **available execution time** among the regions of main storage containing either a ready foreground job or a ready background. Available execution time, in this sense, is the period from the time of the calculation until the next scheduled swap out. Whenever a major time slice expires, the calculation is repeated with the new number of ready regions.

The minor time slice is not quite equivalent to a period of execution time -- a job may have to wait for I/O or some resource during its minor time slice. In this case, control is given to another region until the waiting job is ready again. If it does not become ready before its minor time slice expires, it may wait until the next cycle of minor time slices before executing again.

All terminal jobs are assigned the same dispatching priority, so their Task Control Blocks (TCBs) are grouped together on the queue of active TCBs maintained by the operating system task supervisor. Because the dispatcher always searches this queue from the top when looking for the

next task to receive control, there is an effective priority within the time-sharing TCB group based on the order in which the TCBs are found. The TSO control routines adjust this order to effect the dispatching of a task currently assigned a minor time slice. When the minor time slice of the top foreground job expires, its TCBs are moved to the bottom of the group.

The installation can adjust or weight the fraction of available execution time assigned to each ready region, or it can suppress division of the time altogether. The system operator, or a control user, specifies how many regions are active, and how much execution time, if any, is to be guaranteed to jobs running in the background regions. Three possible methods of calculating the minor time slices, called **simple**, **even**, and **weighted dispatching**, are described in the following paragraphs.

**Simple Dispatching:** In this case, the minor time slice is set equal to the available time, and assigned to the TCB at the top of the time-sharing TCB group (which will always be the job swapped in longest ago). Expressed as an algorithm:

$$MS = AT$$

where MS is the minor time slice and AT is the available execution time, or the time remaining before the next scheduled swap out.

If the operator has requested a percentage of execution time for the background regions, available time is reduced by that amount before the minor time slice is calculated. When the minor time slice expires, in this case before time for a swap out, the remaining time is assigned to TCBs representing jobs in the background regions, in whatever priority they may have. If no background percentage is requested, any background jobs will receive only the execution time that the foreground jobs cannot use.

Simple dispatching is always used whenever only one foreground region is present in the system.

**Even Dispatching:** When more than one foreground region is defined, the installation can specify **even dispatching** of the foreground jobs. In this case, the available execution time is divided evenly among the ready foreground regions.

The algorithm is:

$$MS = \frac{AT}{N}$$

where N is the number of foreground regions containing jobs ready to execute. As in simple dispatching, available time is reduced before the calculation by any guaranteed background percentage.

The first minor time slice is assigned to the foreground job at the top of the group of time-sharing TCBs on the queue. When the minor slice expires, the TCBs associated with that job are moved to the bottom of the time-sharing group, and the next foreground job receives a minor time slice.

**Weighted Dispatching:** The third way the minor time slice calculation can be performed is on a weighted basis. This method allows the system to compensate for jobs that are likely to spend much of their minor time slice in the wait state, usually because of pending I/O requests. (But not for pending terminal I/O, since a job waiting for terminal I/O is not swapped in, and never becomes eligible for a minor time slice.) Under weighted dispatching, the system keeps an **estimated wait time percentage** for terminal job, based on averages of time spent waiting by each job during previous major time slices. Jobs with a high estimated wait time percentage tend to be **I/O-bound**, and will donate much of their time slices to jobs with TCBs on the queue below them. Jobs with low estimated wait time percentages tend to be **compute-bound**, and will use most of their minor time

slice themselves. It is often desirable to assign the I/O-bound job a weighted, or longer, minor time slice to compensate for its "donation" of execution time to other jobs.

To weight the minor time slices, the system forms a sum of the estimated wait time percentage of the jobs to be assigned minor slices in the current cycle. Each job is then given a fraction of the available execution time equal to its fraction of the total estimated wait time percentages.

The algorithm is:

$$MS = \frac{\text{This job's EWT\%}}{\text{Sum of EWT\%s}} \times (AT)$$

where MS is the minor slice to be assigned to a terminal job, EWT% is the estimated wait time percentage, and AT is the available execution time for this minor slice cycle, again adjusted for any guaranteed background percentage.

As an example, consider a minor time slice calculation for two foreground regions, one containing Job A, which is expected to wait 40 percent of its minor time slice; the other containing Job B, which is expected to wait only 10 percent. The sum of the estimated wait time percentages is 50 percent. Job A gets 40/50, or 4/5, of the available execution time as its minor time slice. Job B is assigned 10/50, or 1/5, of the available execution time. However, Job B will probably be able to execute for about 40 percent of Job A's minor time slice too (while Job A is waiting for I/O), and so will end up with under half the available execution time -- about what it would have been assigned on an equal division. Job A, however, will be able to get its I/O started, wait for it to complete, and still have some processing time left to handle the data or issue another I/O request. On an equal division of available time, its minor time slice might have expired before its first I/O request completed.

The estimation of wait time percentage is made by updating a running average of a job's wait time percentages at the end of every major time slice. In making the average, a weighting factor is used to emphasize recent usage over earlier usage. The weighting factor is called the **wait time decay constant**. Its purpose and function is similar to the region activity decay constant, and it can also be specified by the installation. Values appropriate for general job mixes are included in SYS1.PARMLIB.

## System Implementation

This chapter is intended for the programmers and system analysts responsible for generating and maintaining a system with TSO. (The discussions assume that the reader is familiar with the **System Summary** chapter of this publication.) The discussions contain specific information needed to maintain or generate a system with TSO.

For example, the discussion “**Tailoring a Message Control Program**” does not discuss the role a message control program plays in a TSO configuration, but rather provides the syntax and meaning of the macro instructions used to generate a message control program.

This chapter includes discussions of how to:

- Generate (or tailor) a Message Control Program.
- Write the cataloged procedures used by TSO.
- Specify TSO starting parameters.
- Tune the Time Sharing Driver and use TSO Trace.
- Write an installation exit for the SUBMIT command processor.
- Write an installation exit for the STATUS, OUTPUT, and CANCEL command processors.
- Write a LOGON Pre-prompt exit.

### Tailoring a Message Control Program

A Message Control Program, (MCP), handles terminal I/O for TSO. An installation must tailor the MCP to match its needs.

For further information about TCAM and Message Control Programs in general, see **IBM System/360 TCAM Programmer's Guide and Reference Manual, GC30-2024**.

TSO includes a standard Message Control Program (MCP) to handle terminal I/O for those installations that use TSO for all their TCAM applications. An installation tailors a Message Control Program in three steps. First, three macro instructions must be assembled: LINEGRP, LISTTA, and TSOMCP. The output of this assembly is a series of TCAM (Telecommunications Access Method) macro instructions which must, in turn be assembled. The output of this second assembly forms an MCP that must then be linked edited into SYS1.LINKLIB.

#### Mixed Environment MCPs

If an installation requires a mixed environment Message Control Program, because of TCAM applications programs, (message processing programs), it must generate an MCP using TCAM macro instructions instead of the special TSO MCP generating macro instructions. The TCAM macro instructions are used to generate an MCP containing the TSO Message Handler, and any other message handlers for particular terminal applications, and the necessary terminal I/O control blocks.

The communications lines which are to be used with TSO must be dedicated to the TSO Message Handler through the terminal I/O control blocks and the communications lines for TCAM applications must be dedicated to their message handlers.

In addition to the standard TCAM macro instructions, the TSOMH macro instruction can be used to form a TSO Message Handler.

The TSOMH macro instruction has one operand; CUTOFF which specifies a maximum message length. The syntax of the TSOMH macro instruction is:

```
TSOMH  [CUTOFF=integer]
        300 ]
```

**CUTOFF=**

specifies the maximum number of bytes before the remainder of an input message is lost to the system. The value must be an integer between 150 and 65535; the default is 300.

**TSO-Only MCP**

The following is an explanation of each step of the generation of the MCP supplied with TSO:

- Step 1 - Assembly of the one or more LINEGRP macro instructions each followed optionally by one or more LISTTA macro instructions, all followed by the TSOMCP macro instruction. The resultant output is a temporary data set containing assembler language source statements -- TCAM macro instructions which constitute a Message Control Program, that will be used as input to Step 2.
- Step 2 - Assembly of the TCAM MCP macro instructions generated within Step 1. The output of Step 2 is the MCP object module placed into a temporary data set.
- Step 3 - Linkage Editing of the object modules from Step 2 into SYS1.LINKLIB to create an executable MCP load module.

Figure 34 shows the Job Control Language necessary to run these steps.

```

//MCPGEN      JOB      Job card parameters
//STEP1       EXEC     ASMFC
//ASM.SYSPUNCH DD      DSN=%%TCM,DISP=(,PASS),
//              UNIT=SYSDA,SPACE=(CYL,(1,1))
//ASM.SYSIN   DD      *
//              LINEGRP
//              LISTA
//              LINEGRP
//              TSOMCP
//              END

/*
//STEP2       EXEC     ASMFC,COND=(4,LT,STEP1.ASM)
//ASM.SYSPUNCH DD      DSN=%%OBJ,DISP=(,PASS),
//              UNIT=SYSDA,SPACE=(CYL,(1,1))
//ASM.SYSIN   DD      DSN=*.STEP1.ASM.SYSPUNCH,
//              DISP=(OLD,PASS)
//STEP3       EXEC     PGM=LINK EDIT,COND=(4,LT,STEP2.ASM)
//SYSMOD      DD      DSN=SYS1.LINKLIB(IEDQTCAM),DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//SYSUT1      DD      UNIT=SYSDA,SPACE=(1024,(50,20))
//SYSLIB      DD      DSN=SYS1.TELCLIB,DISP=SHR
//SYSLIN      DD      DSN=*.STEP2.ASM.SYSPUNCH,
//              DISP=(OLD,PASS)

```

Figure 34. Job Stream to Tailor MCP

### LINEGRP Macro Instruction

The LINEGRP macro is used to define a **line group**, a group of terminals with similar characteristics: for example, a group of IBM 2741 terminals. The operands of the LINEGRP macro instruction specify:

- The types of terminals in the line group. (TERM)
- The ddname of the DD statements that define the communications lines as data sets. (DDNAME)
- The number of lines, that is, physical device addresses in the line group. (LINENO)
- The number of TCAM basic units, per terminal buffer. (UNITNO)
- The translation tables to be used to translate from the terminal code to EBCDIC. (TRANTAB)
- The character string identifying the transmission code being used when dynamic translation is required. (CODE)
- Whether switched or nonswitched lines are used in this line group. (DIAL)
- The polling interval for polled terminals in this line group. (INTVL)

- The special features the terminals in this line group have -- that is, Transmit or Receive Interruption and for 1050, Text Timeout suppression. (FEATURE)
- The polling and addressing character of terminals in this line group, for 1050 and 2260/2265. (ADDR)
- The screen sizes for IBM 2260 and 2265 Display Stations.

#### LINEGRP Macro Instruction Format

| Name            | Operation  | Operand |
|-----------------|--|---------|
| ( name )LINEGRP | TERM=type<br>DDNAME=ddname<br>LINENO=number<br>[UNITNO=number]<br>[TRANTAB=( table ,table...)]<br>[CODE=(string ,string...)]<br>[DIAL= { YES }<br>{ NO }]<br>[INTVL=number]<br>[FEATURE=( BREAK, ) ( ATTN, ) ( TOSUPPR )<br>( NOBREAK, ) ( NOATTN, )]<br>[ADDR=character string]<br>[SCREEN=( integer, integer)]<br>[TERMNO=( integer, integer)] |         |

#### TERM=

Specifies the type of terminal making up this line group. Only one of the following can be selected:

- 1050 -- defines a line group consisting of IBM 1050 Printer-Keyboards on either switched (dial) or non-switched (direct) lines.
- 2741 -- defines a line group consisting of IBM 2741 Communications Terminals on either switched or non-switched lines.
- 5041 -- defines a line group consisting of both IBM 2741s and IBM 1050s. The terminals in this line group must be on switched (dial) lines.
- 3335 -- defines a line group consisting of Teletype Model 33 or Model 35 terminal or both. The terminals in this line group must be on switched (dial) lines.
- 226L-- defines a line group consisting of IBM 2260 Display Stations connected on a local line.
- 226R-- defines a line group consisting of IBM 2260 Display Stations, connected on a remote line, and optionally IBM 2265 Display Stations.

#### DDNAME=

Specifies the ddnames of the DD statements that define, as a data set, the terminal lines in the line group . These DD statements are found in the cataloged procedure that is used to start the MCP.

#### LINENO=

Specifies the number of lines in this line group. The value must be an integer between 1 and 51.

#### UNITNO=

Specifies the number of basic units per buffer for terminals in this line group. A basic unit is used by TCAM to construct I/O buffers. The default value is 1.

## TRANTAB=

Specifies the translation tables to be used for this line group. If this parameter is omitted, all of the supplied translation tables that are valid for the terminal type specified by TERM= will be included except those marked with an asterisk.

| TERM= | TRANTAB= | Common Name    |
|-------|----------|----------------|
| 1050  | 1050     |                |
| 2741  | CR41     | Correspondence |
|       | EB41     | EBCDIC         |
|       | BC41*    | BCD            |
| 5041  | 1050     | BCD            |
|       | BC41*    | BCD            |
|       | EB41     | EBCDIC         |
|       | CR41     | Correspondence |
| 3335  | TTYB     | TTY parity     |
|       | TYC*     | TTY non-parity |
| 226L  | EBCD     |                |
| 226R  | 2260     |                |
| 2265  | 2265     |                |

\*Not used as a default translation table.

Note: If more than one table is specified explicitly or implied by default, the MCP will use the CODE parameter to determine the proper translation table dynamically.

## CODE=

Specifies the character string use to determine the terminal character set. Each time a terminal is connected, the MCP translates the input line from that terminal using each of the translation tables specified in the TRANTAB operand. The MCP compares the translated result with the character string specified in the CODE= operand. When the MCP finds a match, it uses the appropriate translation table for that terminal from then on.

The default is CODE=LOGON unless the TRANTAB operand specified both BC41 and EB41 (2741 BCD and 2741 EBCDIC). If both EBCDIC and BCD characters sets are present in the line group, the default is CODE="LOGON.

An installation can specify a maximum of four character strings other than LOGON, but each of them must be eight or fewer characters.

## DIAL=

Specifies whether the line group is a dial (switched) line group. If this parameter is omitted, YES is assumed. DIAL=NO is required for TERM=226L, 226R, and 2265.

## INTVL=

Specifies the poll delay intervals in seconds for polled lines. The value should be an integer between 1 and 255. If this parameter is omitted, a value of two is assumed for polled lines. Zero is assumed for 2741, 3335, and 226L terminal types.

**FEATURE=**

Specifies the special features that define this line group:

- BREAK** Specifies that terminals in this line group have the Transmit Interruption feature.
- NOBREAK** Specifies that terminals in this line group do not have the Transmit Interruption feature. This operand should be specified when any of the terminals in the line group do not have the feature.
- ATTN** Specifies terminals in this line group have the Attention feature (Received Interruption.)
- NOATTN** Specifies that terminals in this line group do not have the Attention Feature.
- TOSUPPR** For 1050 terminals, this operand specifies that the optional Text Time-out Suppression feature is present. This operand applies only to 1050 terminals and should be specified only if all 1050 terminals in a 1050 or 5041 group have the feature. When specified read inhibit rather than read commands will be used.

The following table describes the features which may be specified for the 1050, 2741, 5041, 2260 and the 3335 (TWX); where:

- D = Default.
- A = Assumed.
- I = Invalid.
- O = Optional.

| Feature | 1050 | 2741 | 5041 | 3335 | 2260 |
|---------|------|------|------|------|------|
| BREAK   | O    | D    | O    | A    | I    |
| NOBREAK | D    | O    | D    | I    | A    |
| ATTN    | D    | D    | D    | A    | I    |
| NOATTN  | O    | O    | O    | I    | A    |
| TOSUPPR | O    | A    | O*   | A    | I    |

\*TOSUPPR is optional for the 1050 terminals in a 5041 line group. It is assumed for the 2741 terminals in the same 5041 line group.

**ADDR=**

Specifies the station identification character (1050) or the two byte control unit device address (226R,226S) of the terminals in the line group. The character string should be the hexadecimal equivalent of the appropriate transmission code.

Hexadecimal characters should be specified without framing characters. For example if the station identification character is "A", the correct specification is ADDR=E2, the hexadecimal equivalent of the 1050 transmission code for the character "A", not ADDR=C1, the hexadecimal equivalent of the EBCDIC character "A". To find the hexadecimal equivalent of a given character in a specific transmission code, consult the component description publication.

For the 1050, only the station identification character value need be specified; the component selection character values will default to the common polling and addressing values for input and output, respectively. 1050 multidrop is not supported.

This parameter is not valid for TERM=2741 or TERM=3335. This parameter is required for TERM=1050 or 5041. For configurations in which the addressing characters vary among the different terminals in the line group as in 2265, the addressing characters should be specified using LISTTA macro instructions (see below) rather than the LINEGRP macro instruction.

**SCREEN=**

Specifies the screen dimensions of the display station(s) on the line. The first integer specifies the number of rows on the screen. The second integer specifies the number of characters per row. Standard IBM screen sizes are 12x80, 12x40, 6x40, and 15x64. Non-standard sizes will be accepted but a warning will be given. The default for this parameter is 12x80.

**TERMNO=**

Specifies the number of terminals attached to each non-switched line, used with TERM=226R, and 2265. Each subparameter specifies the number of terminals attached to the corresponding relative line within the line group. The relative line numbers are determined by the order in which lines in the line group are defined with DD statements in the cataloged procedure used to start the MCP.

**LISTTA Macro**

The LISTTA macro instruction specifies variations in device address (ADDR) within a line group. One or more LISTTA macro instructions can appear after each LINEGRP macro instruction. Each LISTTA macro instruction modifies one line (RLN) within a line group.

**LISTTA Macro Instruction Format**

| Name | Operation | Operand  |
|------|-----------|--|
| name | LISTTA    | RLN=integer<br>[,ADDR=( chars ,chars...)]<br>[,SCREEN] |

**RLN**

Specifies the relative line number within a line group to which the attributes specified in this macro instruction apply. The relative line numbers are determined by the order in which lines in the line group are defined with DD statements in the cataloged procedure used to start the MCP. For example, RLN=1 refers to the line in the line group defined by the first DD statement in the cataloged procedure.

**ADDR=**

Specifies the alphabetic station identification character (1050) or two byte control unit and device address (2260) of the terminal(s) on this line. One character string must be specified for each terminal on the line. Subparameters must be specified in the order in which polling is to take place. Each character string should be the hexadecimal equivalent of the appropriate transmission code representation for the terminal involved. Hexadecimal characters should be specified without framing characters.

Example: ADDR=(A0A1,A0A2) -- for a 2848 Model 2 with two IBM 2260 Display Stations attached.

For a 1050, only the station identification character value need be specified. 1050 multidrop is not supported.

**SCREEN=**

Specifies the screen dimensions of the display station(s) on the line. The first integer specifies the number of rows on the screen. The second integer specifies the number of characters per row. Standard IBM screen size are 12x80, 12X40, 6x40, and 15x64 non standard sizes will be accepted but a warning will be given. The default for this parameter is (12x80).

**TSOMCP Macro**

The TSOMCP macro instruction:

- Names the MCP, (provides the CSECT name).
- Defines the size of the TCAM basic units used to construct terminal I/O buffers.
- Specifies which TCAM trace tables will be provided.
- Specifies whether a cross-reference table will be included in the MCP.
- Specifies whether the operator can change parameters when he starts the MCP.

**TSOMCP Macro Instruction Format**

| Name                             | Operation | Operand  |
|----------------------------------|-----------|--|
| name                             | TSOMCP    | [UNITSIZ=number]<br>[TRACE=number]<br>[DTRACE=number]<br>[LUNITS=number]<br>[OLTEST=number]<br>[OPTIONS=( XREF , PROMPT )] |
| Note: All operands are optional. |           |  |

**name**

Provides the CSECT label for the generated program. This field is required.

**UNITSIZ=**

Specifies the size of a TCAM basic unit and must be a value between 33 and 255 inclusive. If omitted, the MCP uses a default value of 44. UNITSIZ should be a multiple of 8, plus 4 for efficient main storage usage.

**TRACE=**

Specifies the number of TCAM I/O trace table entries in the Message Control Program. The default value is zero. Maximum value is 65535.

**DTRACE=**

Specifies the number of TCAM Dispatcher Trace Table entries in the Message Control Program. The default value is zero. Maximum value is 65535.

### LNUNITS=

Specifies the number of TCAM basic units to be provided in the buffer pool for creating line buffers for this MCP. A maximum of 65,535 may be specified. If this operand is omitted, the system will calculate a default value using the following algorithm:

#### LNUNITS=

2 x (number of terminals) x (UNITNO value)

or

2.5 x (number of terminals) x UNITNO (for 2265/65)

where:

UNITNO (as specified in each LINEGRP macro) represents the number of units per buffer for terminals defined in the associated line group. If UNITNO is omitted in the LINEGRP macro, the default value (1) is used. This means that each buffer will consist of one basic unit.

If both the LNUNITS and UNITNO keywords are defaulted, the buffer pool created will consist of 2 buffers per terminal with each buffer being one basic unit in length. (PCI buffering is used for both input and output.

### OLTEST=

Specifies the number of 1024 byte blocks used for On-line Test procedures. The value must be between 0 and 255. The default is 0, meaning On-line Test is not used. If online test is used, the value must be from 10 to 255. For a system with 2260 or 2265 display stations the value must be from 14 to 255.

## OPTIONS

### XREF

A cross-reference table including control blocks for each line will be included in the MCP. If this option is omitted, the cross-reference table will be excluded.

### PROMPT

If PROMPT is specified, the system operator will be asked to enter parameters when TCAM is started. At that time he may enter and override some of the parameters specified when the MCP was assembled. The following TCAM parameters are ones which an installation may want to specify when it starts TCAM for TSO. The last parameter entered must be a "U" to end the prompting process. See **IBM System/360 Operating System: TCAM Programmer's Guide and Reference**, GC30-2024 for a description of the INTRO macro instruction and the parameters which can be overridden.

KEYLEN = integer

K = integer

Specifies the size of the basic units, with which the terminal I/O buffers are constructed. This corresponds to UNITSIZ= parameter.

LNUNITS = integer

B = integer

Specifies the number of basic units which are used to build buffers. It corresponds to LNUNITS. The value must be between 0 and 65535.

## STARTUP

= C

S

Specifies that a “cold” start is to be performed following a shutdown of the Message Control Program or a system failure. It is required if `OPTIONS=PROMPT` was specified on the `TSOMCP` macro instruction.

`CROSSRF=integer`

F = integer

Specifies the number of entries in the cross reference table, a debugging aid. If `OPTIONS=XREF` is specified in the `TSO MCP`, one entry will be generated for each line. If the operator specifies fewer entries than there are simultaneously open lines, lines opened after the table is full will have no entries.

`TRACE = integer`

T = integer

Specifies the number of `TCAM I/O` trace entries to be allocated; corresponds to `TRACE=` in the `TSOMCP` macro instruction.

`DTRACE = integer`

A = integer

Specifies the number of entries in the `TCAM Dispatcher Trace Table`, corresponds to `DTRACE=` in the `TSOMCP` macro. The `Dispatcher Trace Table` is a debugging aid that keeps a sequential record of `TCAM` subtasks activated by the `TCAM` dispatcher. One four-word entry is created for each subtask activated. When the end of the table is reached, the table is wrapped around; new entries overlay the oldest entries. Maximum to be specified is 65535: If 0 is specified, the table is not generated.

`OLTEST=number`

O = number

Specifies the number of 1024 byte blocks to be used for `On-line Test` procedures. This parameter corresponds to the `OLTEST` parameter of the `TSOMCP` macro instruction. The default is 0, which indicates that `On-line Test` will not be used. If 0 is not specified the value must be from 10 to 255. For a system with 2260 or 2265 display stations the value must be from 14 to 255.

`CIB = integer`

C = integer

Specifies the maximum number of `Command Input Blocks (CIB)` that can be used at any one time in the `TCAM` subsystem, `CIB`'s are the buffers used to contain operator control messages entered at the system console. The maximum that can be specified is 255. If the operand is omitted, “`CIB=2`” is assumed. At least two `CIB`'s should be specified, since `START` uses one. If an attempt is made to enter an operator control message from the system console, and the number of `CIB`'s specified is already in use, the message is rejected by `TCAM`.

Figures 35 and 36 show the MCP macro specifications for two sample systems.

The first system has:

1. 10 lines for leased, (non-switched), 2741's; all are BCD terminals and use EBCDIC character set only. All terminals in this line group have both Receive and Transmit Interrupt features.
2. 5 lines of teletype (which could be either 33 or 35).
3. The system operator will be prompted to enter TCAM parameters when he starts TCAM. At that time he can override any of the parameters specified on the TSOMCP macro as well as TCAM parameters. See the description of the TSOMCP macro instruction, for parameters pertinent to TSO. (The operator will always have to reply "s=c,u" STARTUP=COLD and a "u" to terminate prompting.) A Dispatcher Subtask Trace Table, useful for debugging purposes, is to be included in the MCP. It will contain 100 4 byte entries. (DTRACE=100)

The sample system shown in Figure 36 has 10 dial lines, to be used by both 1050's and 2741's. The station identification character for the 1050's is "A". Notice that it is specified in terminal transmission code, (E2) not EBCDIC (C1). Assume there are four types of terminals in the line group.

- A. Three 1050's, with Text Timeout Suppression feature and Receive and Transmit Interrupt features.
- B. One 1050, with Text Timeout Suppression feature.
- C. Five 2741's, with Correspondence Code, Receive and Transmit Interrupt features.
- D. Two 2741's, with EBCDIC code.

The default is ATTN and NOBREAK.

Users at terminals in groups A and C could use the TERMINAL command to request Transmit Interrupt handling, (BREAK) or the installation could provide a special LOGON cataloged procedure for these users containing a suitable interruption during output, or while the keyboard is locked, or after a number of consecutive lines of output, when output is being sent. This also could be specified in a LOGON procedure.

```
LINEGRP  TERM=2741,DDNAME=LNGP 2741,LINENO=10,          X
          TRANTAB=EB41,DIAL=NO
LINEGRP  TERM=3335,DDNAME=LNGPTWX,LINENO=5
TSOMCP   OPTIONS=PROMPT,DTRACE=100
```

Figure 35. Sample MCP

```
LINEGRP  TERM=5041,DDNAME=DIAL5041,LINENO=10,ADDR=E2,    X
          FEATURE=TOSUPPR
TSOMCP
```

Figure 36. Sample MCP

## Writing Cataloged Procedures for TSO

Two categories of cataloged procedures are used by TSO. The first includes procedures invoked by the system operator when he starts any of these four TSO tasks:

1. The Message Control Program (MCP).
2. The Time Sharing Control Task (TSO).
3. The Background Reader for the SUBMIT command (BRDR).
4. The TSO Trace Writer.

The second category consists of those procedures invoked each time a LOGON command is entered at a terminal. The PROC operand of the LOGON command specifies the name of the cataloged procedure which:

1. Contains the JCL statements that define the data sets available to the terminal user.
2. Specifies the name of the Terminal Monitor Program (TMP) supplied with TSO or the user-written substitute for the TMP.

Both categories of cataloged procedures must be members of SYS1.PROCLIB or members of partitioned data sets concatenated to SYS1.PROCLIB.

### Message Control Program

The cataloged procedure used to start the Message Control Program specifies through the PGM= operand of the EXEC statement the MCP to be started. The MCP should be named IEDQTCAM. This name allows the MCP to run in a region smaller than MINPART and ensure that the MCP can not be canceled, that is the operator must halt it. Specify ROLL=(NO,NO) to preclude an attempt to Rollout the MCP. Specify DPRTY=(15,15) to insure high priority. The MCP must run at a higher priority than the TSC.

The cataloged procedure used to start the MCP also must define any terminals attached to the system as data sets. This is done through the ddnames specified in the LINEGRP macro instructions used in generating the MCP. Figure 37 shows two procedures that can be used to start the two sample MCPs generated in Figure 35 and 36.

```

//MCP1 EXEC   PGM=IEDQTCAM,ROLL=(NO,NO),TIME=1440,DPRTY=(15,15),REGION=70K
//LNGP2741    DD UNIT=021      FIRST LINE GROUP DATA SET 2741
//           DD UNIT=022
//           DD UNIT=023
//           DD UNIT=024
//           DD UNIT=025
//           DD UNIT=026
//           DD UNIT=027
//           DD UNIT=028
//           DD UNIT=029
//           DD UNIT=02A
//LNGPTWX     DD UNIT=02B      SECOND LINE GROUP DATA SET TWX
//           DD UNIT=02C
//           DD UNIT=02D
//           DD UNIT=02E
//           DD UNIT=02F

//MCP2 EXEC   PGM=IEDQTCAM,ROLL=(NO,NO),TIME=1440,DPRTY=(15,15),REGION=66K
//DIAL5041    DD UNIT=021      LINE GROUP DATA SET
//           DD UNIT=022
//           DD UNIT=023
//           DD UNIT=024
//           DD UNIT=025
//           DD UNIT=026
//           DD UNIT=027
//           DD UNIT=028
//           DD UNIT=029
//           DD UNIT=02A

```

Figure 37. Sample MCP Start Procedures

### Time Sharing Control Task

The cataloged procedure used to start the Time Sharing Control Task contains the Job Control statements defining all the system resources the TSC requires. The procedure consists of an EXEC statement and several Data Definition statements.

The EXEC statement of the cataloged procedure that starts the Time Sharing Control Task, specifies:

- The TSC program name, which is IKJEAT00.
- The TSC region size. This size can be overridden (1) by the TSCREGSZ parameter of the TSO start parameters in SYS1.PARMLIB, (2) by the operator on the START command, or (3) by the TSC initialization routines if a larger region is required as in the case of an installation written Driver.
- ROLL=(NO,NO) to preclude an attempt to Rollout the TSC region, if OPTIONS=ROLLOUT has been specified during system generation.
- DPRTY= to set a priority for the TSC. It must be lower than the MCP.

Five data sets must be defined.

- SYSPARM -- The library containing TSC initialization parameters. These parameters are discussed under "TSO System Parameters".
- SYSUADS -- The User Attributes Data Set, this data set cannot be concatenated.

- **SYSLBC** -- The Broadcast data set which contains messages for users. In addition, the broadcast data set contains a list of valid users, generated by the **ACCOUNT** command and its subcommands.
- **SYSWAP00** -- The swap data sets.
- **IEFPDSI** -- The partitioned data set containing LOGON cataloged procedures. This data set may be either **SYS1.PROCLIB** or a partitioned data set dedicated to LOGON procedures. A dedicated data set will speed up LOGON processing.

For each of these data set definitions, **DISP=SHR** should be specified.

If an installation uses the TSO dump, **SYSTSDP**, the TSO dump data set, usually a tape volume, should be defined.

Figure 38 shows a sample cataloged procedure to start the TSC.

|            |      |   |
|------------|------|---|
| //IEFPROC  | EXEC | PGM=IKJEAT00,ROLL=(NO,NO),DPRTY=(13,13) |
| //SYSPARM  | DD   | DSN=SYS1.PARMLIB,DISP=SHR               |
| //SYSUADS  | DD   | DSN=SYS1.UADS,DISP=SHR                  |
| //SYSLBC   | DD   | DSN=SYS1.BROADCAST,DISP=SHR             |
| //SYSWAP00 | DD   | DSN=SYS1.SWAP1,DISP=SHR                 |
| //SYSWAP01 | DD   | DSN=SYS1.SWAP2,DISP=SHR                 |
| //IEFPDSI  | DD   | DSN=SYS1.PROCLIB,DISP=SHR               |

Figure 38. Sample Cataloged Procedure to Start Time Sharing Control Task

The data definition **ddname** on the **DD** statement defining the **SWAP** data set specifies whether serial or parallel swapping is to be used. The **ddname** is of the form:

**SYSWAPln**

where **l** indicates the level of the data set, i.e., 0 for prime, 1 for first overflow; and **n** is the data set number at this level.

For example, if an installation has two data sets and wants to use parallel swapping it would use **SYSWAP00** and **SYSWAP01** as the **ddnames**.

If an installation wanted to use a **IBM 2301** drum for a prime swap data set and a **IBM 2314** as overflow, the **ddnames** would be **SYSWAP00** for the 2301 the prime data set, and **SYSWAP10** for the 2314, the first overflow data set.

If a system or TSO failure causes TSO to be restarted, you can use **IMDPRDMP** program to save the swap data sets before attempting to restart TSO. When invoking **IMDPRDMP**, the **DD** statements for the swap data sets should be the same as those in the TSO cataloged procedure; the **//PRINTER** **DD** statement writes to tape with chained scheduling and a large blocking factor so that the data sets are dumped quickly. The publication **IBM System/360 Operation System: Service Aids, GC28-6719** shows the procedures for analyzing system failures and how to use the **IMDPRDMP** program to save the swap data sets.

### Starting and Stopping TSO:

When the operator starts TSO for the day, he must:

1. Issue a **START** command to start the Message Control Program. The operand of the **START** command is the name of the cataloged procedure that provides the Job Control statements necessary to execute the MCP. For example if the cataloged procedure used to start the MCP is named **TCAM**, the operator will issue a **START TCAM** command.

2. Issue a **START** command to start the Time Sharing Control Task (TSC). The operand of this command names a cataloged procedure used to start the TSC. For example if the cataloged procedure used to start the TSC is name **TS**, the operator would issue a **START TS** command.

When the operator stops TSO for the day, he must:

1. Issue a **STOP** command to stop the Time Sharing Control Task. The operand of the **STOP** command must be the same as the operand that was used to start the TSC.
2. Issue a **HALT** command to stop the Message Control Program. If the **PGM=** operand of the **EXEC** statement in the cataloged procedure used to start the MCP is **IEDQTCAM**, then the MCP cannot be cancelled with a **CANCEL** command. If the operator cancels the MCP, the TSC must be stopped before the MCP is restarted. The MCP cannot be halted with a **HALT** command unless TSO is stopped.

#### **Defining a UADS using the TSC Procedure:**

When a TSO system is first started after system generation, it is necessary to construct a UADS using the **ACCOUNT** command. The distributed UADS contains one valid user: **IBMUSER** and this user is authorized to use one procedure: **IKJACCNT**. The installation manager should use the **ALLOCATE** command to define a new UADS with a file name of **SYSUADS** and a data set name other than **SYS1.UADS**, specifying a volume serial number. The installation manager should then define its UADS structure with a series of **ACCOUNT** command **ADD** subcommands. He should then log off, stop the system, and change the **SYSUADS DD** statement in the TSC start procedure, to point to the new UADS.

**Note:** The **ACCOUNT** command subcommands in addition to changing the UADS, also maintain a list of valid userids in the Broadcast data set. This list is checked by the **SEND** command before any messages are sent. If an installation generates a new TSO system and saves the old UADS, it must also save the old Broadcast Data Set.

#### **Background Reader (BRDR)**

The cataloged procedure used to start the Background Reader (BRDR) contains Job Control statements that:

- Specify the program name of the Background Reader.
- Pass the Background Reader standard Reader-Interpreter parameters.
- Define required data sets.

The Background Reader, (BRDR), runs as a system task. It is started by the operator. It interprets Job Control Language passed by a terminal user with the **SUBMIT** command. If there is no input for the BRDR, it will relinquish its region and wait for input. Output from the BRDR is placed on **SYS1.SYSJOBQE** and is queued for execution by a standard initiator. The cataloged procedure that provides the Job Control Language to start the Background Reader is similar to other reader procedures. The BRDR program name is **IKJEFF40**. Figure 39 shows an example of a BRDR procedure. For further information on writing system reader/interpreter cataloged procedures, see **IBM System/360 Operating System: MVT Guide, GC28-6720**.

An installation exit can gain access to and modify or delete any JCL passed by the **SUBMIT** command processor. The section, "Writing Installation Exits for the **SUBMIT** Command" describes how to write this exit.

```

//BRDR EXEC PGM=IKHEFF40, X
// REGION=70K, X
// PARM='READERPARM'
//IEFPDSI DD DSN=SYS1.PROCLIB, X
// DISP=SHR
// DD UNIT=SYSDA, X
// SPACE=(80,(500,50),RLSE,CONTIG), X
// DCB=(BUFNO=2,LRECL=80,BLKSIZE=80,DSORG=PS, X
// RECFM=F,BUFL=80)
//IEFRDER DD DUMMY

```

Figure 39. Sample Background Reader (BRDR) Procedure

### TSO Trace Writer

The TSO Trace Writer collects Time Sharing Driver Entry Codes and writes them out to a data set. The Trace Writer operates in its own partition and is started by the operator. A cataloged procedure distributed with TSO defines the resources needed to run TSO Trace.

The cataloged procedure used to start the TSO Trace Writer:

- Specifies the program name of the TSO Trace facility.
- Passes to the Trace Writer a parameter which controls sampling rate.
- Defines the TSO Trace output data set.

Figure 40 shows the procedure. The sample procedure specifies that the Trace Writer output data set is to be written to a 2400 tape unit. The output data set can also reside on disk. The user may specify that chained scheduling be used if trace data set is on tape. If an installation specifies in the DCB operand of the DD statement an NCP value, it must be at least three, that is,

DCB=(BLKSIZE=&BLKSIZE,NCP=3).

An installation should not include a SYSABEND or SYSUDUMP statement in the TSO TRACE cataloged procedure.

```

//TXTRACE PROC TRREGN=20K,          DEFAULTS: REGION SIZE=20K
// TRPARAM=100,                    ENTRY RATE=100 ENTRIES/SEC
// VOLCNT=20,                      VOLUME COUNT=20
// BLKSIZE=2048                   BUFFER SIZE=2048
//* DESCRIPTION OF SYMBOLIC PARAMETERS ---
//* TRREGN - TRACE WRITER REGION SIZE
//* TRPARAM - AN ESTIMATE OF THE RATE AT WHICH ENTRIES WILL BE MADE
//*          INTO TRACE BUFFERS IN NUMBER OF ENTRIES PER SECONDS
//* VOLCNT - MAXIMUM NO. OF VOLUMES AVAILABLE FOR TRACE DATA SET
//*          PER RUN. MAXIMUM VALUE ALLOWED IS 255.
//* BLKSIZE - SIZE OF TRACE BUFFERS. MINIMUM SIZE ALLOWED BY TRACE
//*          WRITER IS 128; MAXIMUM ALLOWED BY SYSTEM IS 32,760
//*
//IEFPROC EXEC PGM=IKJFATRC,        INVOKES INITIALIZATION MODULE
// DPRTY=14,                       PRIORITY SHOULD AT LEAST BE HIGHER
// REGION=&TRREGN.                  THAN CPU-BOUND JOBS IN THE SYSTEM
// PARM=&TRPARAM
//*
//IEFRDER DD DSNAME=TSTRACE,        NAME OF TRACE DATA SET
// UNIT=2400                        DATA SET CREATED ON 9-TRK TAPE(s)
// DISP=(NEW,KEEP),
// DCB=(BLKSIZE=&BLKSIZE),
// VOLUME=( , , , &VOLCNT)

```

Figure 40. Sample TSO Trace Start Procedure

## Logon Cataloged Procedure

The LOGON cataloged procedure defines the system resources that the terminal user can use. The LOGON cataloged procedure can be named in the PROC operand of the LOGON command, supplied through a user exit from the LOGON processor. This procedure:

- Defines or allows for dynamic allocation of all data sets used by the terminal user.
- Specifies which program is to be invoked after LOGON, the TMP distributed with TSO or a user written program.

The data sets defined can include the common system utility data sets, and data sets used by the compilers such as SYSUT1, SYSUT2 or even the specialized data sets used by the Assembler or the Linkage Editor.

In addition any data sets that will be allocated through the ALLOCATE command must have a corresponding DD DYNAM statement. Any data sets needed by a processing program such as a compiler or a system utility can be defined dynamically through the ALLOCATE command or through Dynamic Allocation.

The Terminal Monitor Program distributed with TSO is named IKJEFT01. If a user written TMP is to be used for a particular procedure, then its module name should be substituted for IKJEFT01 in the PGM=operand on the EXEC statement.

The PARM operand on the EXEC statement is interpreted by the Terminal Monitor Program (TMP) as the first line of input from the terminal.

ROLL=(NO,NO) should be specified to preclude rolling out the Time Sharing Region.

REGION= is ignored

The command library, SYS1.CMDLIB, contains the command processor load modules. An installation can also load many of these modules into the TSO Link Pack Area. The command library can be concatenated to SYS1.LINKLIB or defined in the LOGON procedure as a step library.

Note: If the command library is defined in the LOGON procedure as a step library, the modules in the TSO Link Pack Area will not be used. This will degrade performance.

To concatenate SYS1.CMDLIB to SYS1.LINKLIB, use the LNKST00 member of SYS1.PARMLIB. See **IBM System/360 Operating System: MVT Guide, GC28-6720** for further information about using LNKST00.

Figure 41 shows an example of a LOGON procedure.

The sample LOGON procedure can be useful to a programmer using COBOL. Statement 1 specifies the TSO standard TMP for execution. Statement 2 defines the data set containing the HELP command messages. Statement 3 defines a utility data set used by several command processors while statement 4 defines the EDIT utility data set. Statements 5, 6, and 7 define utility data sets used by the COBOL compiler. Statement 8 defines the COBOL subroutine library. Statements 11 through 17 define data sets which can be allocated during the terminal session by the user or a program he invokes, using the ALLOCATE command. Statement 18 defines SYSPROC, an installation defined partitioned data set containing command procedures.

|            |      |  |     |
|------------|------|--|-----|
| //COPROC   | EXEC | PGM=IKJEFT01,ROLL=(NO,NO)                  | 001 |
| //SYSHELP  | DD   | DSN=SYS1.HELP,DISP=SHR                     | 002 |
| //SYSUT1   | DD   | DSN=&SYSUT1,UNIT=SYSDA,SPACE=(CYL,(10,10)) | 003 |
| //SYSEDIT  | DD   | DSN=&EDIT,UNIT=SYSDA,SPACE=(1688,(50,20))  | 004 |
| //SYSUT2   | DD   | DSN=&SYSUT2,UNIT=SYSDA,SPACE=(TRK,(10,5))  | 005 |
| //SYSUT3   | DD   | DSN=SYSUT3,UNIT=SYSDA,SPACE=(TRK,(10,5))   | 006 |
| //SYSUT4   | DD   | DSN=&SYSUT4,UNIT=SYSDA,SPACE=(TRK,(10,5))  | 007 |
| //SYSLIB   | DD   | DSN=SYS1.COBLIB,DISP=SHR                   | 008 |
| //SYSIN    | DD   | TERM=TS                                    | 009 |
| //SYSPRINT | DD   | TERM=TS                                    | 010 |
| //DD1      | DD   | DYNAM                                      | 011 |
| //DD2      | DD   | DYNAM                                      | 012 |
| //DD3      | DD   | DYNAM                                      | 013 |
| //DD4      | DD   | DYNAM                                      | 014 |
| //DD5      | DD   | DYNAM                                      | 015 |
| //DD6      | DD   | DYNAM                                      | 016 |
| //DD7      | DD   | DYNAM                                      | 017 |
| //SYSPROC  | DD   | DSN=CMDPROC,DISP=SHR                       | 018 |

Figure 41. Sample LOGON Cataloged Procedure

## TSO System Parameters

When the Time Sharing Control Task initializes the TSO system, it reads a series of parameters from a member of the partitioned data set named on the SYSPARM DD statement. The SYSPARM DD statement appears in the cataloged procedure used to start the TSC. The member name is IKJPRM00 or a name supplied by the operator on the START command. There are three types of parameters.

- TSC parameters.
- Time Sharing Driver parameters.
- Parameters dealing with the allocation of terminal buffers.

All of these parameters have an effect on the size of the TSC region. The publication **IBM System/360 Operating System: Storage Estimates**, GC28-6551 gives formulas for assessing the effects of these parameters on region size.

### The Time Sharing Control Task Parameters

The TSC parameters:

- Define the number and size of the Time Sharing regions.
- Optionally specify a size for the TSC region.
- Specify the maximum number of users.
- Specify whether SMF is to be used.
- Specify which DRIVER to use.
- Limit the number of tracks the SUBMIT command can use to queue jobs.
- Define the module contents of the Time Sharing Link Pack Extension.

Notes:

- All parameters except LPA and DUMP/NODUMP may be overridden on the START command.
- USERS, SMF, REGSIZE(n), and SUBMIT may be changed by a MODIFY command.
- TERMAX, REGNMAX, and MAP must be specified either in SYS1.PARMLIB or on the START command used to start TSO.

The contents of the Time Sharing Link Pack Area, that part of the TSC region containing reenterable modules common to different TSO applications has a direct effect on system response and overhead. The following routines are used by different users many times during an average session and should reduce loading time if included.

- The I/O Service routines -- that is GETLINE, PUTLINE, PUTGET, and STACK.
- The TMP mainline routines.
- Command Scan -- a service routine used to check the syntax of commands.
- TIME -- a routine used to get the time of day.
- PARSE -- a routine that analyzes the syntax of commands.

In addition if EDIT is being used extensively, portions of the EDIT command processor should be included.

- The Edit Mainline routines.
- INPUT subcommand processor.
- LIST subcommand processor.
- CHANGE subcommand processor.
- Implicit change processor, that is , the update function for portions individual lines.

### Driver Parameters

The DRIVER parameters define dispatching and swap scheduling algorithms (1) for user jobs within one time sharing region and (2) between two time sharing regions.

The DRIVER parameters dealing with activity within each region determine:

- The type of swap scheduling to be used, preemptive or round robin.
- The number of service queues in a region.
- The cutoff points for each queue in terms of swapload or main storage residence between interactions (terminal I/O requests).
- The algorithms used to calculate time slices for each queue.

Certain parameters apply to service queues and require queues to be defined through the SUBQUEUES parameter. These parameters usually are used in pairs, one parameter specifying that a certain criterion is to be used in queue placement and the other specifying the value to be used.

For example, the SWAPLOAD/NOSWAPLOAD parameter specifies whether or not swap load will be used as a criterion for determining which queue a user will be put in. If SWAPLOAD has been specified, the MAXSWAP parameter defines the maximum swap load for each queue.

These paired parameters dealing with service queues are:

SWAPLOAD/MAXSWAP(n,m)=iii

AVGSERVICE/SERVICE(n,m)=iii

OCCUPANCY/MAXOCCUPANCY(n,m)=iii

Note: 'n', 'm', and 'iii' refer to region number, queue number and a value associated with the parameter, such as k byte blocks for SWAPLOAD.

The DRIVER parameters dealing with activity between time sharing regions determine:

- Whether new users are to be assigned to a region based on region activity.
- What type of dispatching the Driver uses between multiple regions or between foreground and background tasks.

There are three types of dispatching: (1) Simple, (2) Even, and (3) Weighted. The parameters involved are PRIORITY, WAIT and BACKGROUND.

If an installation specifies NOPRIORITY and NOWAIT, then simple dispatching is used. The formula used is:

$$MS = AT$$

where MS is the minor time slice and AT is the available execution time, or the time remaining before the next scheduled swap out.

If an installation specifies PRIORITY and NOWAIT, then even dispatching is used. The formula used is:

$$MS = \frac{AT}{N}$$

where N is the number of foreground regions containing jobs ready to execute. As in simple dispatching, available time is reduced before the calculation by any guaranteed background percentage.

If an installation specifies PRIORITY and WAIT, then weighted dispatching is used. The formula used is:

$$MS = \frac{\text{This job's EWT\%}}{\text{Sum of EWT\%s}} \times (AT)$$

where MS is the minor slice to be assigned to a terminal job, EWT% is the estimated wait time percentage, and AT is the available execution time for this minor slice cycle, again adjusted for any guaranteed background percentage.

In a single region system, NOWAIT, NOACTIVITY, and NOPRIORITY should be specified.

The decay constants for the wait estimate (DECAYWAIT) and the activity estimate (DECAYACT) are used to smooth out excessive variations in activity and I/O wait time. If set to 100, (that is, a decay constant of one since the value is in hundredths), will mean that the current value has a weight equal to the total prior value. This will "smooth" out the effects of excessive variations.

The expression used is:

$$\text{old-value} = \frac{(\text{new-value} \times \text{decay-constant}) + \text{old-value}}{(\text{decay-constant} + 1)}$$

MINSLICE, the minimum amount of residence time (major time slice) given to a user program on a given service queue, should be set to allow a useful amount of execution time. One way to calculate MINSLICE is to divide the assumed average number of users for a normal load by the acceptable response time.

If AVGSERVICE has been specified, then the major time slice for users on a given queue is calculated each time the queue is serviced. The value specified by SERVICE for that queue is divided by the number of ready users and the result is compared to the MINSLICE value. The higher of the two is used as the major time slice.

This means that with a normal number of users active, each user program will receive a major time slice at least equal to the MINSLICE value, with a given average response. As the number of users drops, the average response time will remain the same but each user program will receive a longer time slice, based on the SERVICE value.

If PREEMPT is specified, CYCLES should be set to zero, since a higher priority queue will preempt a lower priority queue.

The system operator is prompted for all necessary values that are not specified. For example if OCCUPANCY has been specified, but no MAXOCCUPANCY values entered, the system operator will have to supply values for each service queue in each region. At least one service queue must be defined for each region, and a CYCLES value specified for all service queues defined to preclude unnecessary prompting.

If a parameter is specified or entered incorrectly, the parameter is ignored, but no error message is issued.

### **Buffer Control Parameters**

TSO controls the allocation of terminal buffers in the TSC region. Buffer allocations are based on initial parameters specified in SYS1.PARMLIB.

The BUFSIZE= parameter specifies the size in bytes of each TSO terminal buffer.

The BUFFERS= parameter specifies the total number of TSO buffers. The remaining parameters deal with allocating the number of buffers per user when a given number of users is logged on.

TSO maintains a count of the number of allocated buffers per user, both for input and output. When the number of buffers either for input or output rises to a given level, the user is prevented from continuing until more buffers are available. If the specified maximum number of input buffers are allocated, the keyboard is locked up. If the maximum number of output buffers are allocated, the user's program is put into a wait. This level is determined by the OWAITHI value for output and the INLOCKHI value for input.

When the number of logged on users changes by the percentage specified in the USRCHGE parameter, and when the number of users falls below SLACK value, the number of buffers per user is readjusted. The number of buffers for input and output are distributed in the same ratio as specified by INLOCKHI and OWAITHI.

## System Parameter Format

The format of the parameter records is:

```
  b bparameter-owner b keyword=value b...
```

where b stands for one or more blanks.

The possible parameter-owners are:

- TS -- parameters for the Time Sharing Control Task.
- DRIVER -- parameters for the Time Sharing Driver.
- TIOC -- parameters controlling terminal buffer allocation.

Keywords cannot be continued but may be repeated. This has the effect of continuation, as repeated keyword values are added on to those already specified. When two parameters conflict, the last value is used. Figure 42 shows an example of system parameters for a single region model 50 and for a double region model 65. Figure 43 shows the syntax and meaning of the start parameters.

| PARAMETER OWNER | KEYWORD   | MEANING  |
|-----------------|---|--|
| TS              | TERMAX=nnnn   | Specifies maximum number of terminals the installation wants to support. Must be less than 10000.  |
|                 | REGNMAX=nn  | Specifies the maximum number of TSO user regions. Must be between 1 and 14 inclusive.  |
|                 | MAP=nnnn  | Specifies the number of entries in the User Main Storage Map for each user. Entry describes area to be swapped. Used to reduce swapping of unused storage.   |
|                 | USERS=nnnn  | Specifies the initial maximum number of users the system will allow to log on. Must be between 0 and the value specified on TERMAX. If nnnn greater than TERMAX, TERMAX value will be used. Defaults to TERMAX.  |
|                 | SMF= ( $\left[ \begin{array}{l} \text{OFF} \\ \text{OPT}=1 \\ \text{OPT}=2 \end{array} \right] \left[ \text{, EXT}=\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} \right] \right)$ | Standard SMF foreground parameters, see <b>System Management Facilities</b> , GC28-6712.   |
|                 | DSPCH=cccccc  | Specifies first six characters of Time Sharing Driver. Defines names of all four driver modules. Last two characters must be 00 to 03. Defaults to IKJEAD, the driver supplied with TSO.   |
|                 | LPA=(module list)   | List of modules to be included in Time Sharing Link Pack Extension.  |
|                 | REGSIZE(n)=<br>(nnnnnK,xxxxxK)  | Specifies the time sharing region number and size of that region. n is the region number and nnnnnK its size. xxxxxK is size of Local System Queue Area (LSQA). LSQA must be smaller than region size but greater than zero. n must be between 1 and REGNMAX. nnnnn and xxxxx are number of contiguous 1024 byte areas wanted, should be even, and their sum may range from 0 to 16382. Odd numbers specified will be rounded up to next higher even number. |
|                 | SUBMIT=nnn  | Specifies maximum number of tracks in SUBMIT command job queue. Defaults to limit set at system generation.  |
|                 | TSCREGSZ=nnnnnK   | Specifies amount of main storage to be allocated to Time Sharing Control Task region. nnnnn is number of contiguous 1024 byte areas desired, must be even, and may not be more than 16382. An odd number will be rounded up to next higher even number. If not specified in either SYS1.PARMLIB or in START command, Time Sharing Control Task will calculate its own region size.   |
|                 | DUMP= DUMP<br>NODUMP  | DUMP indicates that swap units are to be marked reserved. Necessary if a SWAP dump to be taken.  |
| TIOC            | BUFSIZE=nn  | Specifies size of terminal buffer. Default 44.   |
|                 | BUFFERS=nn  | Total number of buffers.   |
|                 | OWAITHI=nn  | Specifies the maximum number of allocated output terminal buffers per user in order to put a user program into output wait.  |
|                 | INLOCKHI=nn   | Specifies the maximum number of allocated input terminal buffers per user in order to lock a users keyboard.   |
|                 | OWAITLO=nn  | Specifies the number of allocated output buffers to bring a user out of output wait state. In other words if OWAITLO=4, when 4 or less buffers remain allocated, the user is brought out of output wait.   |

Figure 42. TSO System Parameter Syntax (Part 1 of 4)

| PARAMETER<br>OWNER | KEYWORD     | MEANING  |
|--------------------|-------------|--|
|                    | INLOCKLO=nn | Specifies the number of currently allocated input buffers to unlock the terminal keyboard for input. In other words, when the number of allocated input buffers fail to or below the INLOCKLO value, the users's keyboard is unlocked. |
|                    | USERCHG=nn  | Specifies percentage of change in logged on users needed to redistribute buffers and recalculate the OWAITHI and INLOCKHI numbers during slack time.   |
|                    | RESVBUF=nn  | Specifies the total number of terminal buffers that must be free to avoid locking all terminals to prevent input.  |
|                    | SLACK=nn    | Specifies number of logged on users that constitute slack time.  |

DRIVER PARAMETERS

| PARAMETER        | EFFECT   | OPERAND   |
|------------------|--|---|
| ACTIVITY         | Use average region activity in selecting a region for a new user. Single region systems should specify NOACTIVITY.   | none  |
| AVGSERVICE       | Calculate major time slice for users on a queue by dividing number of ready users by value set in SERVICE.   | none  |
| BACKGROUND=nn    | The specified percentage of available CPU time is guaranteed for background (non-TSO) tasks.   | nn<br>percentage  |
| CYCLES(n,m)=iiii | The number of times an entire queue will be serviced (ie. each ready user swapped in before the next lower queue is serviced.) If zero is specified; the queue is serviced until empty, (no ready users) Cycles must be specified for each region. | n<br>region<br>m<br>service<br>queue<br>iiii<br>cycles  |
| DECAYACT=nnnn    | The specified exponential decay constant (in 1/100ths) will be used to smooth average region activity. Higher values make history more important.  | nnn<br>exponential<br>decay constant<br>in 1/100ths.<br>Value of 100<br>makes history<br>as important<br>as current<br>value. |
| DECAYWAIT=nnnn   | The specified exponential decay constant (in 1/100ths) will be used to smooth average I/O wait time. Higher values make history more important.  | nnn<br>exponential<br>decay constant<br>in 1/100ths.<br>Value of 100<br>makes history<br>as important<br>as current<br>value. |

Figure 42. TSO System Parameter Syntax (Part 2 of 4)

| PARAMETER              | EFFECT   | OPERAND  |
|------------------------|--|--|
| MAXSWAP(n,m)=iii       | A user maximum swap load for queue m in region n is iii 1024 byte blocks. A user exceeding this swap load will be moved to a lower priority queue.   | n<br>region<br>m<br>service queue<br>iii<br>1024 byte<br>blocks of<br>swap load. |
| MAXOCCUPANCY(n,m)=iiii | Occupancy is the accumulated time a user has resided in main storage since the last terminal interaction. If a user on queue n in region m, accumulates more than iii 1/100ths of a second of occupancy, he will be moved to a lower queue.                        | n<br>region<br>m<br>service queue<br>iiii<br>1/100ths<br>seconds.                |
| MINSLICE(n,m)=iiii     | The minimum amount of core residency allotted to a user on queue m in region n is iii (1/100ths) of a second.  | n<br>region<br>mmm<br>number of<br>service queues                                |
| NOACTIVITY             | Region activity average will not be used in assigning new users to regions. NOACTIVITY should be specified for single region systems. Default of ACTIVITY.   | none   |
| NOAVGSERVICE           | AVGSERVICE calculations are not performed. Default of AVGSERVICE.  | none   |
| NOBACKGROUND           | Background tasks will not be guaranteed a minimum amount of execution time. Default of BACKGROUND.   | none   |
| NOOCCUPANCY            | The length of time a user program is resident will not be used to position him on a service queue. Default of OCCUPANCY.   | none   |
| NOPREEMPT              | Preemptive swap scheduling is not used. Default of PREEMPT.  | none   |
| NOPRIORITY             | Minor time slices are not calculated Default or PRIORITY.  | none   |
| NOSWAPLOAD             | Swap load (size of programs) will not be criterion for service queue placement. Default of SWAPLOAD.   | none   |
| NOWAIT                 | I/O WAIT time estimate will not be used in dispatching between multiple regions. Causes even dispatching between regions if PRIORITY was specified, simple dispatching if NOPRIORITY was specified. Should be specified for single region system. Default of WAIT. | none   |

Figure 42. TSO System Parameter Syntax (Part 3 of 4)

| PARAMETER          | EFFECT  | OPERAND  |
|--------------------|---|--|
| OCCUPANCY          | Residence times since last terminal I/O interaction are accumulated for current user program to use for service queue placement.  | none   |
| PREEMPT            | Preemptive swap scheduling is used. Assumes multiple service queues defined through SUBQUEUEES parameter. A user who comes ready on a higher queue preempts a resident user on a lower queue. The preempted user is swapped out after a minslice of the preemptor has elapsed. The preemptors entire queue is serviced. | none   |
| PRIORITY           | If NO WAIT was specified, each region receives an equal minor time slice. If WAIT was specified, regions receive minor time slices based on I/O WAIT time.  | none   |
| SERVICE(n,m)=iiii  | The amount of time (iiii 1/100ths of seconds) is divided by the number of ready users in queue m, region n. The result is compared with MINSLICE, and the larger is used as the major time slice. This calculation is performed whenever a queue is to be cycled, (all the ready users swapped in).                     | n<br>version<br>m<br>queue<br>iiii<br>1/100ths<br>of seconds |
| SUBQUEUEES (n)=mmm | The number of service queues in region 'n' is 'm'. Lower service queue numbers have higher priority. If only one service queue is defined, then round robin swap scheduling is used. At least one service queue must be defined for each region.  | n<br>region<br>mmm<br>service<br>queue                       |
| SWAPLOAD           | Swap load or size of program in 1024 byte blocks, is used for queue placement.  |  |
| WAIT               | Causes I/O wait time estimate to be calculated for region. Used with weighted dispatching.  |  |

Figure 42. TSO System Parameter Syntax (Part 4 of 4)

```

TS   TERMAX=10  REGNMAX=1  REGSIZE(1)=(100K,8K)
TS   LPA=(IKJPTGT,IKJSCAN,IKJEF02,IKJEFT25)
TS   LPA=(IKJPARS)
DRIVER   AVGSERVICE  PREEMPT  SUBQUEUEES(1)=3
DRIVER   CYCLES(1,1)=0
DRIVER   CYCLES(1,2)=0
DRIVER   CYCLES(1,3)=0
DRIVER   MAXOCCUPANCY(1,1)=750  MINSLICE(1,1)=150
DRIVER   MAXOCCUPANCY(1,2)=1500  MINSLICE(1,2)=750
DRIVER   MAXOCCUPANCY(1,3)=4500  MINSLICE(1,3)=4500
DRIVER   SERVICE(1,1)=150
DRIVER   SERVICE(1,2)=1500
DRIVER   SERVICE(1,3)=6000
TIOC   BUFSIZE=44
TIOC   BUFFERS=80
TIOC   OWAITHI=8
TIOC   OWAITHI=4
TIOC   INLOCKHI=4
TIOC   INLOCKLO=2
TIOC   SLACK=01
TIOC   RESVBUF=10
TIOC   USERCHG=99

TS   TERMAX=60  REGNMAX=2  REGSIZE(1)=100K,8K)  REGSIZE(2)=(100K,8K)
TS   LPA=(IKJPTGT,IKJSCAN,IKJEFT02,IKJEFT25)
TS   LPA=(IKJEBEM4,IKJEBELP,IKJEBELT,IKJEBECH,IKJEBELI)
TS   LPA=(IKJPARS)
DRIVER   WAIT
DRIVER   ACTIVITY
DRIVER   OCCUPANCY
DRIVER   AVGSERVICE
DRIVER   PREEMPT
DRIVER   DECAYWAIT=100
DRIVER   DECAYACT=100
DRIVER   SUBQUEUEES(1)=3  SUBQUEUEES(2)=3
DRIVER   CYCLES(1,1)=0  CYCLES(1,2)=0  CYCLES(1,3)=0
DRIVER   CYCLES(2,1)=0  CYCLES(2,2)=0  CYCLES(2,3)=0
DRIVER   MAXOCCUPANCY(1,1)=500  MINSLICE(1,1)=100
DRIVER   MAXOCCUPANCY(1,2)=1000  MINSLICE(1,2)=500
DRIVER   MAXOCCUPANCY(1,3)=3000  MINSLICE(1,3)=3000
DRIVER   MAXOCCUPANCY(2,1)=500  MINSLICE(2,1)=100
DRIVER   MAXOCCUPANCY(2,2)=1000  MINSLICE(2,2)=5000
DRIVER   MAXOCCUPANCY(2,3)=3000  MINSLICE(2,3)=3000
DRIVER   SERVICE(1,1)=1000
DRIVER   SERVICE(1,2)=1000
DRIVER   SERVICE(1,3)=6000
DRIVER   SERVICE(2,1)=100
DRIVER   SERVICE(2,2)=1000
DRIVER   SERVICE(2,3)=6000
TIOC   BUFSIZE=44
TIOC   BUFFERS=300
TIOC   OWAITHI=8
TIOC   OWAITLO=4
TIOC   INLOCKHI=4
TIOC   INLOCKLO=2
TIOC   SLACK=12
TIOC   RESVBUF=60
TIOC   USERCHG=01

```

Figure 43. Sample TSO System Parameters

## Tuning a System With TSO

This section describes some techniques useful in improving the performance of a system with TSO. Some of these techniques act directly upon the underlying MVT control program, but they all affect TSO system performance. The specific objectives discussed are:

- Reducing I/O contention between foreground (TSO) and background tasks.
- Reducing time spent fetching modules.
- Reducing swap overhead that is, the amount of I/O performed copying foreground job images between main and secondary storage.
- Reducing seek time between different libraries and different parts of the same library.

The techniques are discussed in terms of their objectives. Figure 44 summarizes the various techniques in terms of the times at which they are used.

|   | SYSGEN   | SYS1.PARMLIB<br>Construction                 | Writing<br>Cataloged<br>Procedures | Writing<br>LOGON<br>Procedures<br>UADS<br>Building | Library<br>Positioning                               |
|---|--|--|------------------------------------|--|--|
| Reducing<br>Foreground/<br>Background I/O<br>Contention | UNITNAME<br>IOREQ=PRIORITY                                     |  | UNIT                               | UNIT   |  |
| Reducing<br>Fetch Time                                  | Operator Communications<br>Residence Options<br>Transient Area | BLDL (SVC)<br>BLDL (LINKLIB)<br>TSLPA<br>LPA |                                    |  |  |
| Reducing<br>Seek Time                                   |  |  |                                    |  | ABSTR (JCL)<br>IEHDASDR (VTOC)<br>IEBCOPY<br>IEHMOVE |
| Tuning the<br>Driver                                    |  | TSO TRACE                                    |                                    |  |  |

Figure 44. Summary of Tuning Objectives for Various Techniques

### Reducing I/O Contention Between Foreground and Background

To reduce the I/O contention between foreground (TSO) tasks and background tasks:

1. During System Generation, use the UNITNAME macro instruction (1) to separate direct access space into foreground (TSO) and background areas and (2) to define generic names, (unitnames), for these spaces. (System Generation GC28-6554 discusses the UNITNAME macro instruction.)

In cataloged procedures for background use, define utility data sets with the UNIT parameter specifying one of the background unitnames, (for example, SYSDA).

In cataloged procedures for foreground use, notably the LOGON procedures, define utility data sets using a UNIT parameter specifying one of the foreground (TSO) unitnames, (for example, SYSTS).

Restrict placement of data sets allocated dynamically, (using the ALLOCATE command), by specifying a foreground unitname in the UADS. Define a unit for dynamically allocated data sets in the UADS with the ADD subcommand of the ACCOUNT command.

Be sure to give the PUBLIC attribute, in the PRESRES member of SYS1.PARMLIB, to at least one of the Foreground Unitname volumes so that SYSOUT data sets will be dynamically allocated on a specific unit. See MVT Guide, GC28-6720, for more information on the PRESRES member of SYS1.PARMLIB.

2. Specify priority scheduling on I/O for the direct access devices on which TSO data sets reside. Since foreground jobs usually run at higher priority than background jobs, this should reduce the time Foreground (TSO) jobs wait for background I/O requests. Use the IOQUEUE operand of the IODEVICE macro instruction. For further information, see System Generation GC28-6554.

For an example of the use of UNITNAME, if your system has two 2314 Direct Access Storage Facilities available, at addresses 130 through 137 and 230 through 237, you could define generic names for scratch, utility, and spooling data sets by using the UNITNAME macro instructions specified in Figure 45.

```
UNITNAME UNIT=( 130,4 ),( 230,4 ),NAME=SYSDA
* BACKGROUND SCRATCH UNITNAME, NON-TSO
UNITNAME UNIT=( 134,4 ),( 234,4 ),NAME=SYSTS
* FOREGROUND TSO UNITNAME
```

Figure 45. Sample Use of the UNITNAME Macro Instruction

For the system in this example, if a data set is defined with a UNIT parameter specifying SYSDA, it will reside on one of the 2314 packs at addresses 130 to 133 or 230 to 233. Data sets defined with a UNIT parameter specifying SYSTS, will reside on one of the 2314 packs at addresses 134, 135, 136, 137, 234, 235, 236, 237, or 237.

There are advantages to using generic naming through the UNITNAME macro facility:

1. There is no arm contention between foreground and background utility data sets.
2. TSO utility data sets can be demounted easily when TSO is not active.

If the UADS specifies that new dynamically allocated data sets should reside on the units in the foreground unitname group, then there will be no arm contention between these data sets and background data sets.

### Reducing Fetch Time

Reducing the time spent fetching modules from libraries is one of the most important controllable factors in system performance. For the purposes of this discussion, fetch time is defined as the time span from the start of execution of a macro instruction which obtains a module from secondary storage and optionally transfers control to the module to completion of this operation. Examples of these types of macro instructions are LINK and LOAD.

Note: The following list of modules for residence and directory residence were selected based on a count of modules fetched during TSO operation. The modules are included because they fulfill at least one of the following criterion:

- High fetch count.
- Reduce seek time between one or more of the following - SVCLIB, VTOC, or CATALOG.
- Required for each line of terminal output.
- Required for each line of terminal I/O.
- Must be refreshed after SWAP-in of not resident.
- Part of Catalog sequence.
- BSAM CHECK routine for terminal I/O.
- High use count for resident access modules. (Increases swap load if not resident.)

If the referenced module is resident in the Link Pack Area, or the TSO Link Pack Area Extension, then no I/O is involved in a fetch. If the module is not resident, but if there is a BLDL entry for it, then a directory search is not required. If there is no BLDL entry, then I/O is required for the BLDL (directory search) and for the actual loading of the module. An installation therefore can influence fetch time through the following four options:

1. SVC module residence.

Figure 46 shows a list of SVC modules and their functions. For further information about loading modules into the Link Pack Area, see MVT Guide, GC28-6720.

2. Resident directory entries (BLDL) for SVC library.

The following Dynamic Allocation modules should be included in the SVC BLDL list:

IGC0109I, IGC0209I, IGC0309I, IGC0709I, IGC0909I IGC1009I, IGC1109I, IGC1209I,  
IGC1309I, IGC1409I, IGC1409I, IGC1509I, IGC1609I, IGC1709I, IGC2609I, IGC2709I,  
IGC2909I.

The following Catalog Modules should be included in the SVC BLDL list:

IGG0CLC2, IGG0CLC3, IGG0CLC7, IGG0CLF2.

3. Resident directory entries (BLDL) for Linkage library, and TSO Command library.

A LINKLIB BLDL entry in main storage occupies 56 bytes. Modules which have BLDL entries have their fetch time significantly reduced. For this reason, it is recommended that the entire list of modules in *Storage Estimates*, GC28-6551, listed under "Reentrant Load Modules that can be made resident in the Time Sharing Link Pack Area," be included in the Linkage Library BLDL list.

4. SVC transient area pairs.

In addition to defining module and directory (BLDL) residence, specifying during system generation two SVC transient areas for each initiator (background region) or each foreground region, decreases total time spent waiting for the loading of a non-resident SVC function.

| Module   | Name/Function                 |
|----------|-------------------------------|
| IGC0002F | SVC 26 CATALOG                |
| ICG0002H | SVC 28 OPEN Extent Catalog    |
| IGC0003C | SVC 33 I/O Halt               |
| IGC0006+ | SVC 60 STAE                   |
| IGC0009+ | SVC 90 XQMNGR                 |
| IGC0009C | SVC 93 TGET/TPUT              |
| IGC0009F | SVC 96 STAX                   |
| IGC0009G | SVC 97 TEST                   |
| IGC0009I | SVC 99 Dynamic Allocation     |
| IGC2509I | Dynamic Allocation Update DSE |
| IGC2509I | Dynamic Allocation Initiation |
| IGG0CLC1 | Catalog Locate                |
| IGG0CLC6 | Catalog Return                |
| IGG09301 | TPUT                          |

Figure 46. SVC Modules useful for Residence in TSO

### Reducing Swap Overhead

There are two ways to reduce swap load:

1. Use the MAP parameter of the TSC start parameters to reduce the amount of unused core swapped. MAP=6 is a recommended value. For further information about the MAP parameter, see "TSO System Parameters."
2. Any TSO modules resident in the TSLPA will not be swapped. This means that TSO modules common to foreground jobs, such as the TMP, should be considered for residency.

### Reducing Seek Time Between Different Libraries and Different Parts of the Same Library

You can reduce seek time and increase I/O overlap by properly placing system data sets on direct access devices.

Use the IEBCOPY utility program to move system data sets. Use the IEHMOVE utility program to move the system catalog. Use the IEHDASDR utility to place the VTOC of a volume. For further information about these utilities, see *IBM System/360 Operating System; Utilities*, GC28-6586.

Use the ABSTR operand of the SPACE parameter on dd statements to position data sets. In arranging libraries on direct access devices, the following points should be kept in mind.

1. Every reference to the system catalog requires a reference to the VTOC, since the catalog must be opened. If the system catalog and the VTOC share cylinders (split cylinders), then seek time is reduced.
2. Unless the CATLG/UNCATLG modules (IGG0CLC2, IGG0CLC3, IGG0CLC7) are resident in the Link Pack Area, they must be fetched during any catalog operation. If these modules are not resident, it is useful to have the system catalog and SVCLIB adjacent on the storage device.
3. To allow for maximum overlap during fetch I/O, either CMDLIB or LINKLIB should reside on a different volume than SVCLIB. Since CMDLIB and LINKIB should be concatenated, they also should probably not reside on the same volume.
4. In general, heavily referenced data sets, like LINKLIB and SYSJOBQE should be in proximity when they share a volume.

The following example shows the dd statements used to define system data sets. The space parameters apply for 2314 volumes. It is assumed that the VTOC was placed on tracks 50 to 52 on these 2314 packs. Cataloging of data sets should be done after this allocation has been run.

```
//FORTLIB      DD  DSN=SYS1.FORTLIB,SPACE=(ABSTR,(28,2,40)),UNIT=2314,      X
//              VOL=SER=SYSRES,DISP=(NEW,KEEP),                          X
//              DCB=(RECFM=U,BLKSIZE=3625)
//CMDLIB       DD  DSN=SYS1.CMDLIB,SPACE=(ABSTR,(20,30,30)),UNIT=2314,      X
//              VOL=SER=SYSRES,DISP=(NEW,KEEP),                          X
//              DCB=(RECFM=U,BLKSIZE=3625)
//SYSCTLG      DD  DSN=SYS1.SYSCTLG,SPACE=(ABSTR,(7,53)),UNIT=2314,        X
//              VOL=SER=SYSRES,DISP=(NEW,KEEP)
//SVCLIB       DD  DSN=SYS1.SVCLIB,SPACE=(ABSTR,(80,60,75)),UNIT=2314,      X
//              VOL=SER=SYSRES,DISP=(NEW,KEEP),                          X
//              DCB=(RECFM=U,BLKSIZE=1024,DSORG=POU)
//PROCLIB      DD  DSN=SYS1.PROCLIB,SPACE=(ABSTR,(20,140,9)),UNIT=2314,     X
//              VOL=SER=SYSRES,DISP=(NEW,KEEP),                          X
//              DCB=(RECFM=FB,BLKSIZE=400,LRECL=80)
//BROADCAST    DD  DSN=SYS1.BROADCAST,SPACE=(ABSTR,(20,30)),UNIT=2314,     X
//              VOL=SER=SYSRS2,DISP=(NEW,KEEP)
//UADS         DD  DSN=SYS1.UADS,SPACE=(ABSTR,(20,53)),UNIT=2314,          X
//              VOL=SER=SYSRS2,DISP=(NEW,KEEP),                          X
//              DCB=(DSORG=PD,RECFM=FB,BLKSIZE=800)
//LOGON        DD  DSN=SYS1.LOGON,SPACE=(ABSTR,(7,73,5)),UNIT=2314,        X
//              VOL=SER=SYSRS2,DISP=(NEW,KEEP),                          X
//              DCB=(RECFM=FB,BLKSIZE=400,LRECL=80)
//JOBQJE       DD  DSN=SYS1.SYSJOBQE,SPACE=(ABSTR,(60,80),,CONTIG),        X
//              VOL=SER=SYSRS2,DISP=(NEW,KEEP),UNIT=2314
//LINKLIB      DD  DSN=SYS1.LINKLIB,SPACE=(ABSTR,(75,140,100)),            X
//              VOL=SER=SYSRS2,DISP=(NEW,KEEP),UNIT=2314,              X
//              DCB=(RECFM=U,BLKSIZE=3625)
```

### Using TSO Trace

The TSO Trace Data Set Processor is a problem program that dumps the output data set from TSO Trace and produces a formatted listing. Figure 47 shows the job control language required to run the TSO Trace Data Set Processor. The example assumes that the TSO Trace Data set has been written to a tape volume with a volume serial number of TTRACE. The listing shows the parameters specified, and provides an explanation of each entry record as well as the contents of the record in hexadecimal and EBCDIC: The contents of register 1 is listed in the third column of the Trace Data Set Processor.

```
//TTRDUMP      JOB  ,MSGLEVEL=1
//STEP        EXEC PGM=IKJFATRP,PARM='CODES=STD'
//SYSPRINT    DD  SYSOUT=A
//TRACEDD     DD  DSN=TTRACE,VOL=SCR=TTRACE,UNIT=2400
```

Figure 47. Sample Job System to Run TSO Trace Data Set Processor

TSO TRACE is a started task which operates in its own region. All Driver Entry Codes are recorded in buffers which are then written to a data set. This data set can be listed by the TSO Trace Data Set Processor or can be analyzed by a user written program. The section of this publication **Writing Cataloged Procedures for TSO**, discusses how to define the TS Trace data set and specify parameters required by TSO Trace. Figure 48 shows the format of the TSO Trace data set.

| Entry Type | When Produced   | Description of Contents   |
|------------|---|---|
| 'A'        | When the trace writer is started                                      | Word 1 X'FFFFFFFD'<br>Word 2 # of 3-word entries per record<br>Word 3 Time of Day in timer units  |
| 'B'        | When the trace writer is stopped.                                     | Word 1 X'FFFFFFFE'<br>Word 2 Date in packed decimal 00YYDDDS<br>Word 3 Time of Day in timer units   |
| 'C'        | When information was lost (volume switching, low sampling rate, etc.) | Word 1 X'FFFFFFF'<br>Word 2 Number of entries lost<br>Word 3 Time of Day in timer units of the first lost entry   |
| 'D'        | Normal entry (contains words 1-3 of the DPA).                         | Word 1 Bytes 1-2 TJID or 0<br>Byte 3 Reserved (X'00')<br>Byte 4 Entry code<br>Word 2 Contents of register 1 on entry to TSIP<br>Word 3 Time of Day in timer units |
| 'E'        | Following a normal entry with entry code 0 (TMP entry).               | Words 1-2 Command name<br>Word 3 Unpredictable  |
| 'F'        | Following a normal entry with entry code 25 (LOGON establishes PSCB). | Bytes 1-7 USERID<br>Bytes 8-12 Unpredictable  |
| 'G'        | Following a normal entry with entry code 44 (FE Serviceability)       | Diagnostic data (There will be 2 <sup>n+1</sup> 3-word groups of data available. The value of n is contained in bits 5-7 of word 2 of the normal entry.           |

Figure 48. Format of the TS Trace Data Set

The TSEVENT macro instruction is issued by system tasks to request services of the Driver or to notify the Driver of specific events. The TSEVENT macro instruction specifies an event name that is translated into a Driver Entry Code. Based on parameters specified to the Driver and on the sequence of these codes, the Driver initiates various actions.

Appendix C lists all the possible event names, the codes they generate, their meanings, and which task issues these codes. Associated with most TSEVENT macro calls is a TJID, which identifies the user to the Driver. The TJID is assigned when the user logs on.

The PARM value on the EXEC statement specifies what entries will be listed. All "G" type records will be listed regardless of the parameters. The individual keyword parameters should be enclosed in apostrophes and separated by commas. The keyword parameters and their syntax are:

#### CODES

specifies which class of entry codes are to be included in the listing. The subparameters, S,T and D represent 'System' codes, 'Terminal I/O' codes, and "Dispatcher" codes, respectively. The listing, therefore, will contain only those entry codes belonging to the class, or classes, specified. Appendix C lists the Entry Code classes. These subparameters may be written in any order, but must not contain delimiters nor embedded blanks. If the CODES parameter is omitted, all non-dispatcher entries will be listed, i.e., CODES=ST is the default option.

**TJID=XXX[- YYY]**

specifies that only entries associated with the TJID specified by the number XXX are to be listed. If YYY is also given, all entries associated with TJID's in the range XXX to YYY, inclusive, are listed. If the value given for XXX is zero, all entries will be listed. (This is also the default if the 'TJID' parameter is not specified.) Both numbers XXX and YYY must be specified as decimal digits. The maximum length of each number is three digits.

**CLOCK=XXXXXXXX[ - YYYYYYYY]**

indicates that no entry before time XXXXXXXX (relative to the starting time of the first entry) is to be included in the listing. If - YYYYYYYY is specified no entry after that time is listed. Both numbers must be specified as decimal digits and given the time in seconds. The maximum length of each number is seven digits.

## Writing Installation Exits for the Submit Command

A user exit from the SUBMIT command allows an installation to:

- Verify a jobname.
- Verify a userid.
- Send a message to the terminal and optionally request a reply.
- Cancel a SUBMIT request.

The TSO SUBMIT command allows a terminal user to initiate a background job. A description of the syntax and use of the SUBMIT command is found in **IBM System/360 Operating System: Time Sharing Option, Command Language Reference, GC28-6732**.

The SUBMIT command processor writes the contents of a user specified data set consisting of Job Control Language statements, (JCL), and input data, onto a logical extension of SYS1.SYSJOBQE. The size of this extension is limited at system generation time by the SUBMITQ operand of the TSO OPTION macro. Size can be further limited by the SUBMIT parameter which the Time Sharing Control Task reads from SYS1.PARMLIB when the operator issues a START TS command.

Any authorized terminal user can submit a background job, but no jobs will be scheduled if the operator has not issued a START BRDR command.

An installation can control foreground initiated background jobs through an installation written SUBMIT exit routine. Through the routine an installation can:

- Delete, modify, or insert statements.
- Request that a message be displayed at the terminal and optionally request a reply.

The routine must be linkage edited as an independent module, given the name IKJEFF10, and cataloged in SYS1.LINKLIB. The SUBMIT command processor invokes the user written exit when the first JOB statement is read. Return codes in register 15 control subsequent calls. The return codes are:

- 0 - continue -- that is process the current statement and read the next.
- 4 - reinvoke the exit for another statement -- that is process the current statement and invoke the exit for the next statement.
- 8 - display a message at the terminal and invoke the exit.
- 12- display a message at the terminal, obtain a response, and invoke the exit. (If the user has specified NOPROMPT, this will cause the SUBMIT processor to abort.)
- 16- abort.

Upon entry to the user written exit routine, register 1 contains the address of a list of six fullwords.

1st word - address of the current statement.

If zero, entry is to get a statement (return code from previous call was 4). To delete the current statement, zero out the first word.

2nd word - address of a message to be displayed on terminal.

If non-zero, return code from previous call was 8 or 12. The exit may free the buffer. If zero, no message, the return code was 0, 4, or this is the first call.

3rd word - address of response.

If the exit return code from the previous call was 12, SUBMIT will free the buffer. The format of both the message and the response is LLtext where LL is a two byte length field containing one length of the text, maximum length 82 bytes.

4th word - address of USERID.

The USERID is 8 characters left justified padded with blanks.

5th word - address of control switches.

Byte 0 specifies under what conditions SUBMIT will call the exit.

| Byte | Bit | Meaning           |
|------|-----|-------------------|
| 0    | 0   | Call for JOB card |
|      | 1   | Exec              |
|      | 2   | DD                |
|      | 3   | Command           |
|      | 4   | Null              |
|      | 5   | Reserved          |
|      | 6   | Reserved          |
|      | 7   | Reserved          |

Byte 1 if non-zero contains the card column where the operand field begins. For example, if the operand field begins in column 16, byte 1 contains hex 10.

Byte 2 specifies what the current statement is.

| Byte | Bit | Meaning                   |
|------|-----|---------------------------|
| 2    | 0   | JOB statement             |
|      | 1   | EXEC                      |
|      | 2   | DD                        |
|      | 3   | command                   |
|      | 4   | null                      |
|      | 5   | operand to be continued   |
|      | 6   | statement to be continued |
|      | 7   | statement continuation    |

If bit 5 is on, bit 6 must be on, but bit 6 can be on and bit 5 off.

Byte 3 is unused.

6th word - for exit's use.

The first time SUBMIT calls the exit, the 6th word is initialized to zeros. The exit can use the word for counters or switches. The value is not changed between calls.

## Writing Installation Exits for the OUTPUT, STATUS, and CANCEL Commands

An installation can write a user exit for the OUTPUT, CANCEL, and STATUS commands. The exit routine is common to all three command processors and is named IKJEFF53. An TSO supplied module performs jobname verification if a user exit is not supplied. The parameters and the return codes have the same format and meaning for all three command processors. The user exit determines which command processor is invoking it from a parameter. The parameters are passed through a standard linkage with register one containing the address of a list of seven fullwords.

Word 1 -- contains the address of the jobname.

Word 2 -- contains the address of the length of the jobname.

Word 3 -- contains the address of the userid.

Word 4 -- contains the address of the length of the userid.

Word 5 -- contains the address of a message to be issued to the terminal user. The format of one message is LLtext where LL is a two byte field containing the length of the entire message, maximum length 82 bytes. If 0, the exit is being entered to create a message.

Word 6 -- contains the address of a response from the terminal user. The format of the response is LLtext where LL is a two byte field containing the length of the entire message, maximum length 82 bytes.

Word 7 -- contains the address of the command code.

Command codes are:

0 = STATUS command.

4 = CANCEL command.

8 = OUTPUT command.

Return codes are passed in register 15 and are defined as:

0 = Valid job name, get next job name and continue processing.

4 = Display message, get response, and call exit again. If the terminal use has specified NOPROMPT on his LOGON or PROFILE command, the command will abort and a message will be issued to the terminal.

8 = Display message and call exit again.

12= Invalid jobname, cancel request for foreground initiated background service.

16= Abort (Same action for return code greater than 16.)

## Writing a Logon Pre-Prompt Exit

A user-written exit, cataloged in SYS1.LINLIB can specify most of the values to be determined from the LOGON command or from prompting by the LOGON command processor. These include:

- The userid.
- The password.
- An account character string -- that is the value specified in the ACCT operand.
- A procedure name -- that is the name of a cataloged procedure usually specified in the PROC operand.
- A region size.
- A series of 80 byte card images of Job Control Language (JCL) to be used instead of the JOB and EXEC statements normally constructed by the LOGON processor.
- Portions of the Protected Step Control Block.
- The contents of the User Profile Table.
- The contents of the Environment Control Table used by the LOGON Prompter.

In addition, the exit can:

- Read but not change the Event Control Block which will be posted if the exit terminates due to a CANCEL request.
- Read but not change the completion code from the last step executed from the terminal logging on.

The parameters passed are defined in the PL/I procedure in Figure 49. The variables declared as either BIT or CHAR, VARYING are passed as String Dope Vectors. For a definition of String Dope Vectors see *IBM System/360 Operating System: PL/I-F Programmer's Guide*, GC28-6594. The exit may be written in any language but since parameters are passed as String Dope Vectors, they can be manipulated directly in PL/I. The exit must be Linkage Edited and cataloged in SYS1.LINKLIB with a entry point name which processes standard Operating System parameters and the module must be named IKJEFLD.

The exit receives control as a problem program and can use the I/O service routines through Assembler Language macro instruction (PUTLINE, GETLINE, PUTGET, STAX).

LOGON passes 16 parameters to the user exit. They are of three types:

1. Character String defined in PL/I as CHAR VARYING.
2. Bit Strings defined in PL/I as BIT VARYING.
3. Fullwords defined in PL/I as BINARY FIXED (31).

The parameters passed can be given any name in the user written exit procedure but their meaning is determined by the order in which they appear. The following explanation of the parameters uses the names defined in the PL/I procedure in Figure 49.

**CONTROL SWITCHES** -- a bit string that specifies what actions the exit has taken. The various bit switches are:

**UADS FAIL** -- if this bit is equal to one, on entry to the pre-prompt exit, then there was an unsuccessful ENQ on the UADS entry for the specified userid.

**REGION FAIL** -- if this bit is equal to one on entry to the pre-prompt exit, the region size specified in the LOGON REGION operand was too large to be satisfied. The exit can specify a different region size.

**FAIL** -- if this bit is equal to one on entry to the pre-prompt exit, the LOGON processor will cancel the attempted log on. No message will be issued to the terminal user, so the pre-prompt exit must issue any needed message.

**DONT PROMPT** -- if this bit is equal to one on return from the procedure, the LOGON processor will not prompt the terminal user for any necessary LOGON operand values but will use the values specified by the pre-prompt exit. These include:

- Userid.
- Password.
- Accounting string.
- Procedure name.
- Region size.

**EXIT UADS** -- if this bit equals one on return from the pre-prompt exit, the LOGON processor will not reference the UADS but will take all character strings and bit strings from the procedure. DON'T PROMPT must be set to one if this bit is set to one.

**EXIT JCL** -- if this bit is equal to one on return from the pre-prompt exit, the pre-prompt exit has supplied Job Control Language (JCL) that is to be used instead of the JOB and EXEC statements constructed normally by the LOGON processor.

**EXIT PSCB** -- if this bit is equal to one on return from the pre-prompt exit, the LOGON processor will use the PSCB accounting string returned by the user but will not write it to the UADS at LOGOFF time.

**EXIT ATTR1** -- if this bit is equal to one on return from the pre-prompt exit, the LOGON processor will use the PSCBATR1 string provided by the exit and will not write it into the UADS at LOGOFF time.

**EXIT ATTR2** -- if this bit is equal to one on return from the pre-prompt exit, the LOGON processor will use the PSCBATR2 string returned by the pre-prompt exit and will not write it into the UADS at LOGOFF time.

**EXIT GROUP** -- if this bit is equal to one on return from the pre-prompt exit, the LOGON processor will use the PSCBGPNM string returned by the exit procedure, but will not write it to the UADS at LOGOFF time.

**EXIT UPT** -- if this bit is equal to one on exit from the pre-prompt exit, the LOGON processor will use the UPT string returned by the exit procedure, but will not be written to the UADS at LOGOFF time.

**NO ENQ UADS** -- if this bit equals one and the DONT PROMPT and EXIT UADS bits are both one, the LOGON processor will not ENQ on the UADS entry for the specified user.

If both DONT PROMPT and EXIT UADS are equal to one then:

- EXIT PSCB
- EXIT ATTR1
- EXIT ATTR2
- EXIT GROUP
- EXIT UPT

also must be equal to one.

TERMINAL INPUT LINE -- this parameter contains the first line entered from the terminal.

The values for the next five parameters must be specified if the DONT PROMPT bit is set on one.

USERID -- used to return a userid to the LOGON processor.

PASSWORD -- used to return a password to the LOGON processor.

ACCOUNT -- used to return an accounting string to the LOGON processor.

PROCEDURE -- used to return the name of a cataloged procedure containing JCL to define the resources needed by the terminal job.

REGION SIZE -- used to return to the LOGON processor a region size for the terminal job.

JCL -- used to provide Job Control statements that define terminal job resources instead the JOB and EXEC statement constructed by the LOGON processor.

The next six parameters must have values specified by the pre-prompt exit if EXIT UADS is set on one by the pre-prompt exit.

PSCB -- used by the exit procedure to set a value for the PSCB accounting string.

FIRST ATTRIBUTE -- used to return a value for the PSCBATR1 string.

SECOND ATTRIBUTE -- used to return a value for the PSCBATR2 string.

GENERIC GROUP -- used to return a value for the PSCBGPNM.

UPT -- used to return a value for the UPT.

ECT -- used to return a value for the Environment Control Table (ECT) used by the LOGON prompter.

The last two parameters cannot be altered by the pre-prompt exit but may be read.

ECB -- the Event Control Block (ECB) for the exit procedure.

COMPLETION CODE -- this fullword contains the completion code for the last job step of the last job executed from this terminal.

For the format of the Protected Step Control Block (PSCB), the User Profile Table (UPT), and the Environment Control Table (ECT) see the publication **IBM System/360 Operating System: System Control Blocks**.

```

USER EXIT:  PROCEDURE
            ( CONTROL_SWITCHES |
              TERMINAL_INPUT_LINE,
              USERID,
              PASSWORD,
              PASSWORD,
              ACCOUNT,
              PROCEDURE,
              REGION_SIZE,
              JCL,
              PSCB,
              FIRST_ATTRIBUTES,
              SECOND_ATTRIBUTE,
              GENERIC_GROUP,
              UPT,
              ECT,
              ECB,COMPLETION_CODE ) ;

DECLARE
CONTROL_SWITCHES BIT (*) VARYING,
  UADS_FALL      BIT (1) DEFINED CONTROL_SWITCHES POSITION (1),
  REGION_FAIL    BIT (1) DEFINED CONTROL_SWITCHES POSITION (2),
  CANCEL         BIT (1) DEFINED CONTROL_SWITCHES POSITION (3),
  DONT_PROMPT   BIT (1) DEFINED CONTROL_SWITCHES POSITION (4),
  EXIT_UADS      BIT (1) DEFINED CONTROL_SWITCHES POSITION (5),
  EXIT_JCL       BIT (1) DEFINED CONTROL_SWITCHES POSITION (6),
  EXIT_PSCB      BIT (1) DEFINED CONTROL_SWITCHES POSITION (7),
  EXIT_ATTR1     BIT (1) DEFINED CONTROL_SWITCHES POSITION (8),
  EXIT_ATTR2     BIT (1) DEFINED CONTROL_SWITCHES POSITION (9),
  EXIT_GROUP     BIT (1) DEFINED CONTROL_SWITCHES POSITION (10),
  EXIT_UPT       BIT (1) DEFINED CONTROL_SWITCHES POSITION (11),
  NO_ENQ_USERID BIT (1) DEFINED CONTROL_SWITCHES POSITION (12);

DECLARE TERMINAL_INPUT_LINE CHAR (*) VARYING;
DECLARE USERID              CHAR (*) VARYING;
DECLARE PASSWORD            CHAR (*) VARYING;
DECLARE ACCOUNT             CHAR (*) VARYING;
DECLARE PROCEDURE           CHAR (*) VARYING;
DECLARE REGION_SIZE        BINARY FIXED (31);
DECLARE JCL                 CHAR (*) VARYING;
DECLARE PSCB                BIT (*) VARYING;
DECLARE FIRST_ATTRIBUTE     BIT (*) VARYING;
DECLARE SECOND_ATTRIBUTE    BIT (*) VARYING;
DECLARE GENERIC_GROUP       CHAR (*) VARYING;
DECLARE UPT                 BIT (*) VARYING;
DECLARE ECT                 BIT (*) VARYING;
DECLARE CP_ABEND            BIT (1) DEFINED ECT POSITION (1);
DECLARE CP_RETURN-CODE      BIT (24) DEFINED ECT POSITION (8);
DECLARE IO_WORD_AREA_ADDR   BIT (32) DEFINED ECT
                           POSITION (33);

DECLARE NOSEC_LEVEL_MSG     BIT (1) DEFINED ECT POSITION (65);
DECLARE SEC_LEVEL_MSG_ADDR BIT (24) DEFINED ECT POSITION (73);
DECLARE COMMAND_NAME       CHAR (8) DEFINED ECT POSITION (97);
DECLARE SUBCOMMAND_NAME    CHAR (8) DEFINED ECT POSITION (161);
DECLARE NO_MAIL_SWITCH     BIT (1) DEFINED ECT POSITION (228);
DECLARE NO_NOTICE_SWITCH   BIT (1) DEFINED ECT POSITION (229);
DECLARE ECB                BINARY FIXED (31);
DECLARE COMPLETION_CODE     BINARY FIXED (31);

```

Figure 49. Portion of Sample PL/I Logon Pre-Prompt Exit

## Storage Estimates

The estimates included in this chapter are intended for planning purposes only. None of these estimates have been verified, and they are subject to change. Verified estimates appear in the publication **IBM System/360 Operating System: Storage Estimates, GC28-6551**

This chapter contains two sections: main storage requirements and auxiliary storage considerations. All figures in this chapter are decimal, and “K” represents a factor of 1024.

### Main Storage Requirements

The main storage requirement for TSO is divided into four major parts:

- An addition to the MVT basic fixed requirement.
- The TCAM Message Control Program requirement.
- The Time Sharing Control region requirement.
- The foreground regions in which user’s programs are executed.

Only the first of these requirements has any effect on the batch environment if time sharing is not active. Storage for the TCAM, Time Sharing Control, and foreground regions is obtained from the dynamic area when the operator starts time-sharing operations. This storage is returned to the dynamic area when time sharing is stopped, and is again available for batch processing.

#### *MVT Basic Fixed Requirement*

The main storage basic fixed requirement for an MVT system is for:

- The nucleus.
- The Master Scheduler Region.
- The Link Pack Area (LPA).
- The System Queue Area (SQA).

Storage for the basic fixed requirement is allocated by the Nucleus Initialization Program (NIP) when the system is started and does not normally vary while the system is running.

#### **Nucleus**

Including TSO at system generation adds approximately 3K to the size of the resident MVT nucleus, for a total requirement of about 45K. In addition, communication lines, like other I/O devices, require 40 bytes each in the nucleus for control blocks.

#### **Master Scheduler Region**

The master scheduler region is increased by approximately 4K to handle new or extended operator commands for the time-sharing environment, and for extended error recovery. The total requirement is about 16K.

## **Link Pack Area**

One small TSO module is added to the required MVT link pack area list of resident modules. The minimum link pack area size remains 10K. If the standard MVT resident reenterable load module and resident SVC lists are used at system generation, the LPA requirement is about 54K. If space is available, an additional 16K of SVC modules for time sharing are appropriate for the resident list, for a total LPA size of 70K.

Additional resident reenterable load modules for time sharing are placed in an extension to the link pack area allocated in the Time Sharing Control region, and are resident only when time sharing is active. The size of this extension, called the Time Sharing Link Pack Area (TSLPA), is discussed with the Time Sharing Control Region requirement.

## **System Queue Area**

During time-sharing operations, use of the system queue area is kept to a minimum by placing as many control blocks as possible into a local system queue area (LSQA) defined in each foreground region. Control blocks in the local SQA are swapped in and out of main storage along with the foreground job they apply to.

Some control blocks associated with foreground jobs, such as queue elements for named data sets and operator reply queue elements, must remain in main storage while the job is swapped out. Space for these control blocks, and for all control blocks associated with the tasks supervising the time-sharing operation must be allocated from the system queue area. These requirements must be considered when setting SQA size at system generation or at nucleus initialization.

## ***Message Control Program Requirement***

The size of the TCAM Message Control Program region depends largely on what options are selected and what hardware is present on the teleprocessing network. In addition to the minimum requirement for the Message Control Program routines, there are requirements for each defined line group, each additional terminal type, and for each permitted user. If teleprocessing applications other than TSO are present, additional routines to handle different buffering and queuing techniques will be needed.

In a system with TSO as the only teleprocessing application, with three terminal types and two line groups, the Message Control Program requirement is expected to be about 52K plus 800 bytes for each possible concurrent user. Although the Message Control Program executes in a problem program region, the region may be smaller than the normal minimum problem program region size (MINPART).

## ***Time Sharing Control Region Requirement***

The Time Sharing Control region must provide space for programs for the Time Sharing Control Task, Region Control Tasks, several resident SVC routines, the time sharing extension to the link pack area, and various control blocks. Some of the control blocks are repeated for each foreground region, for each swap data set, or for each time sharing user. An initialization routine brought in when the operator starts time sharing analyzes the time-sharing parameters supplied by the installation, calculates the region size requirement, and obtains the region from the dynamic area.

Using a buffer length of 40 bytes, and assuming eight buffers per time-sharing user, a TSO configuration with two IBM 2314 swap data sets, one foreground region, and 20 users would require a time sharing control region of about 87K. A larger configuration, with two 2301 swap data

sets and two 2314 swap data sets, four foreground regions, and 100 users would require about 117K for the time sharing control region.

### ***Dynamic Area Requirements***

The SEND operator command, like several others already in the MVT configuration, obtains and uses an 12K operator command region from the dynamic area when the operator enters it. This area is freed when processing of the command is completed.

When it is active, the time sharing trace facility requires a 20K region from the dynamic area.

### ***Foreground Region Requirement***

The foreground region contains the programs invoked by the terminal user. Space must be provided in the foreground region for the local system queue area (LSQA) and for four main storage subpools used for control blocks for the command system.

The subpools defined are:

- Subpool 0--4K.
- Subpool 1--4K.
- Subpool 78--2K.
- Subpool 251--2K.

The minimum foreground region size is 72K, and all IBM-supplied command processors except some of the language processors can execute in this region.

### **Auxiliary Storage Requirements**

The major additions to the system auxiliary storage requirements for TSO are for the swap data sets and new or larger system libraries and data sets. The installation must also consider the direct access storage needs of the individual terminal users, and make allowances for these in the size of the system catalog and password data sets. In addition, data sets which are dynamically allocated, use SPACE=(50,10,10) for size parameters.

### ***Swap Data Sets***

A swap data set is divided into swap allocation units, each of which consists of a device-dependent number of 2K records. To avoid space fragmentation, space in the swap data set is always assigned in integral swap allocation units. Figure 50 shows the sizes of allocation units for various swap devices.

| Device | Type                  | Allocation Unit | Size |
|--------|-----------------------|-----------------|------|
| 2301   | Drum Storage          | 1 track         | 18K  |
| 2303   | Drum Storage          | 4 tracks        | 18K  |
| 2305-1 | Fixed Head Storage    | 4 tracks        | 44K  |
| 2305-2 | Fixed Head Storage    | 4 tracks        | 52K  |
| 2314   | Direct Access Storage | 1/2 cylinder    | 64K  |
| 3330   | Disk Storage          | 3 tracks        | 32K  |

Figure 50. Swap Allocation Unit Sizes

For a system with one foreground region, the **maximum** necessary swap space can be calculated by the algorithm:

$$\text{Swap Space} = (R/A) \cdot (U+2)$$

where:

R is the size of the region.

A is the size of an allocation unit, as shown in Figure 50, (R/A is rounded up to an integer).

U is the number of concurrent foreground jobs.

For instance, a system with one foreground region of 120K, an IBM 2314 swap device, and 30 possible users would have a maximum swap data set space requirement of:

$$(120/64) \cdot (30+2) = 2 \cdot 32 = 64 \text{ allocation unit or 32 cylinders}$$

In this case, the number of allocation units required to hold a complete foreground region is two, and the number of users plus two is 32.

Swap data sets must be allocated by cylinder, even if the swap data set does not occupy a full cylinder.

If TSO runs out of swap space, no message is issued, and the system may loop, so allow sufficient space.

### ***System Libraries and Data Sets***

The additions to system libraries for TSO are expected to be (with the increments expressed in 2311 tracks):

- SYS1.LINKLIB--30 tracks.
- SYS1.SVCLIB--20 tracks.
- SYS1.MACLIB--60 tracks.

Two new libraries, SYS1.COMDLIB (command library) and SYS1.HELPLIB (HELP data set), are expected to be smaller than 220 IBM 2311 tracks each.

The size of the User Attribute Data Set, a partitioned data set with a member for each user identification, depends on the number of password-identification-account number-procedure name combinations defined for each user. A simple identification structure for a single user with a single value at each level requires about 200 bytes of storage space.

Typical time-sharing usage also requires more space for the system catalog and password data sets than batch usage. All user data sets are cataloged as a default, and read-only password protection is recommended at least for system data sets. This type of protection does not cause any performance degradation when the data sets are accessed for reading.

## Appendix A: TSO Commands

The commands available to terminal users of the Time Sharing Option are listed below, grouped according to function. Installations may give other names to these commands by assigning aliases to the respective members in the system command library. No IBM-supplied command names include numerals, allowing installations to ensure uniqueness in locally named commands.

### Data Management

#### ALLOCATE

define and allocate a new or old data set.

#### ATTRIB

build a list of attributes that can be assigned to data sets being allocated.

#### CONVERT

convert source programs written in Code and Go FORTRAN or Interactive PL/I to standard format FORTRAN or PL/I.

#### COPY

duplicate a sequential or partitioned data set, or a member of a partitioned data set, optionally modifying such characteristics as blocking factor.<sup>1</sup>

#### DELETE

delete and uncatalog one or more data sets or members.

#### EDIT

invoke the edit mode or input mode to modify or create a data set; provide an interface to the language syntax checkers and processors.

#### FORMAT

format a data set for printing according to embedded controls.<sup>1</sup>

#### FREE

release a data set or an attribute list.

#### LIST

display at the terminal all or part of one or more data sets, optionally re-arranging information in the records.<sup>1</sup>

#### LISTALC

display at the terminal the names and characteristics of currently active (allocated) data sets.

#### LISTBC

display at their terminal any system notices or messages from other users.

#### LISTCAT

display at the terminal the names and characteristics of a group of data sets indexed together in the system catalog.

#### LISTDS

display at the terminal the characteristics of one or more specified data sets.

#### MERGE

copy all or part of one data set or member into another.<sup>1</sup>

---

<sup>1</sup>IBM Program Products. See Appendix B.

**PROTECT**

assign or modify password protection to a data set.

**RENAME**

change the name of a data set or member, or assign an alias to a member.

**Language Processors****ASM**

invoke the prompter for<sup>1</sup> the Assembler (F).

**CALC**

invoke the Interactive PL/I processor for desk calculator mode/<sup>1</sup>

**COBOL**

invoke the American National Standard COBOL compiler.<sup>1</sup>

**FORT**

invoke the FORTRAN (G1) compiler.<sup>1</sup>

**RUN BASIC**

invoke the ITF: BASIC compiler and execution control routines.<sup>1</sup>

**RUN GOFORT**

invoke the Code and Go FORTRAN compiler and execution control routines.<sup>1</sup>

**RUN IPLI**

invoke the ITF: PL/I compiler and execution control routines.<sup>1</sup>

**RUN PLI CHECK**

invoke the PL/I Checkout Compiler.<sup>1</sup>

**RUN PLI OPT**

invoke the PL/I Optimizing Compiler, the OS Loader, and give control to the resulting module.

**PLI**

invoke the PL/I Optimizing compiler.<sup>1</sup>

**PLIC**

invoke the PL/I Checkout Compiler.<sup>1</sup>

**Program Control****CALL**

invoke a specified program which exists in load module form.

**LINK**

invoke the Linkage Editor to create a load module from one or more object and load modules.

**LOADGO**

invoke the Loader to process a specified object module, bring it into storage, and give it control.

**RUN**

invoke a user program in source program form, first compiling it, then calling the Loader to bring it into storage and give it control.

**TEST**

control the execution of a program, interrupting it at pre-specified points for debugging activity.

---

<sup>1</sup>IBM Program Products. See Appendix B.

### **Remote Job Entry**

**Note:** Use of these commands requires authorization in the user profile.

#### **CANCEL**

cancel a job previously submitted for background execution.

#### **OUTPUT**

direct SYSOUT data sets and system messages from submitted jobs to the terminal or a specified data set.

#### **STATUS**

display information at the terminal on the status of a job previously submitted for background execution.

#### **SUBMIT**

submit a data set containing job control language for one or more jobs for interpretation and execution in the background.

### **System Control**

**Note:** Use of these commands requires authorization in the user profile.

#### **ACCOUNT**

add or modify user profiles in the User Attribute Data Set.

#### **OPERATOR**

invoke the operator mode, allowing the user to enter system commands from his terminal.

### **Session Control**

#### **EXEC**

invoke a command procedure.

#### **HELP**

display at the terminal information on command function and syntax.

#### **LOGON**

start a terminal session.

#### **LOGOFF**

end a terminal session.

#### **PROFILE**

specify special characters for line editing; lock out and accept messages from other users.

#### **SEND**

direct a message to the system operator or to another user.

#### **TERMINAL**

specify the conditions under which an attention interruption is to be simulated, for terminals without attention keys; and define other terminal-dependent characteristics.

#### **TIME**

display at the terminal the amount of time expended during the current session or the current program.

## Appendix B: Program Products

The following is a list of the IBM Program Products available for use with TSO. Program Products are available from IBM for a license fee. Program Product Design Objectives for each of the Program Products are available from your local IBM representative.

- **Interactive Terminal Facility (ITF): PL/I and BASIC.**  
A problem-solving language processor. See the following publications: **IBM System/360 Operating System Time Sharing Option: Interactive Terminal Facility: PL/I and BASIC Design Objectives**, GC28-6822. **ITF: PL/I General Information**, GC28-6827. **ITF: BASIC General Information**, GC28-6828.
- **Code and Go FORTRAN.**  
A FORTRAN compiler designed for a fast compile-execute sequence. See the publications: **Code and Go FORTRAN Design Objectives**, GC28-6823. **FORTRAN Program Products for OS and OS with TSO, General Information**, GC28-6824.
- **FORTRAN IV (G1).**  
A version of FORTRAN IV providing specific support for the terminal environment. See the publications: **FORTRAN IV (G1) Processor Design Objectives**, GC28-6845. **FORTRAN Program Products for OS and OS with TSO, General Information**, GC28-6824.
- **TSO FORTRAN Prompter.**  
An initialization routine to prompt the user for options, and invoke the FORTRAN IV (G1) Processor. See the publications: **TSO FORTRAN Prompter Design Objectives**, GC28-6843. **FORTRAN Program Products for OS and OS with TSO, General Information**, GC28-6824.
- **FORTRAN IV Library (Mod 1).**  
Execution-time routines for list-directed I/O, PAUSE, and STOP capability, for use with either Code and Go FORTRAN or FORTRAN IV (G1). See the publications: **FORTRAN INV Library (Mod 1) Design Objectives**, GC28-6844. **FORTRAN Program Products for OS and OS with TSO, General Information**, GC28-6824.
- **American National Standard Full COBOL Version 3.**  
A version of the American National Standard COBOL compiler modified for the terminal environment. See the publications: **American National Standard (ANS) Full COBOL Compiler Version 3 Design Objectives**, GC28-6406.
- **TSO COBOL Prompter.**  
An initialization routine to prompt the user for options, and invoke the American National Standard Full COBOL Version 3 Compiler. See the publications: **TSO COBOL Prompter Design Objectives**, GC28-6404.
- **PL/I Optimizing Compiler.**  
A PL/I compiler designed for compilation of efficient object programs, incorporating a prompter routine allowing invocation from the terminal. See the publications: **PL/I Optimizing Compiler Design Objective**, GC33-0013. **PL/I Optimizing Compiler, General Information**, GC33-0001.
- **PL/I Checkout Compiler.**  
A PL/I compiler designed to simplify the task of testing and debugging programs, incorporating a prompter routine allowing invocation from the terminal. See the publications: **PL/I Checkout Compiler Design Objectives**, GC33-0028. **PL/I Checkout Compiler, General Information**, GC33-0003.

- **OS PL/I Resident Library.**  
A subroutine library for use during the linkage editing of programs produced by the PL/I Optimizing Compiler. See the publication: **OS PL/I Resident Library Design Objectives, GC33-0014.**
- **OS PL/I Transient Library.**  
A subroutine library for use during the execution of programs produced by the PL/I Optimizing and Checkout Compilers. See the publication: **OS PL/I Transient Library Design Objectives, GC33-0015.**
- **TSO Assembler Prompter.**  
An initialization routine to prompt the user for options and invoke the Assembler (F). See the publication: **TSO Assembler Prompter Design Objectives, GC26-3734.**
- **TSO Data Utilities: COPY, FORMAT, LIST, MERGE.**  
A set of commands and EDIT subcommands to manipulate data sets and format text. See the publication: **TSO Data Utilities: COPY, FORMAT, LIST, MERGE Design Objectives, GC28-6750.**

## Appendix C: Driver Entry Codes

Entry Code Table (Part 1 of 4)

| Event Name<br>(Entry Code) | Calling Routine (CLASS)                      | Reason for Entry   | Input      |   |
|----------------------------|--|--|------------|---|
|                            |  |  | Register 0 | Register 1  |
| PPMODE (0)                 | Problem Program (S)                          | Command about to be processed  |            | Address of 8-character command name.<br><br>Bit 0<br>0 - Ended<br>1 - Beginning |
| TSLICE (1)                 | TSC Timer Exit Routine (S)                   | Time slice has expired   | 0          | NA  |
| TERMWAIT (2)               | TGET/TPUT (T)                                | User is waiting for terminal I/O. Swap him out.                                | TJID       | Bit 0<br>0: input<br>1: output<br><br>Bytes 3 and 4: number of free buffers.    |
| NIOWAIT (3)                | Region Control Tasks (S)                     | All user's tasks are in non-I/O Wait.  | TJID       | NA  |
| USERRDY (4)                | DEQUEUE, Terminal Handler, TSLIH, WTOR (S,T) | Swapped out user ready to run.   | TJID       | NA  |
| RUSRTRMW (5)               | Region Control Task (S)                      | Restored user is still in non-I/O Wait.<br><br>Swap him out.                   | TJID       | NA  |
| REQSTNC (6)                | Enqueue (S)                                  | User is setting must complete for owned resources.                             |            | Estimated must complete time.   |
| RELMC (7)                  | Dequeue (S)                                  | User is no longer in must complete status.                                     |            |   |
| DISPLAC (8)                | TS Dispatcher (D)                            | A task switch has taken place resulting in a background task being dispatched. |            |   |
| DISPSYS (9)                | TS Dispatcher (D)                            | A task switch has taken place resulting in a system task being dispatched.     |            |   |

Entry Code Table (Part 2 of 4)

| Event Name<br>(Entry Code) | Calling Routine (CLASS)              | Reason for Entry   | Input      |  |
|----------------------------|--------------------------------------|--|------------|--|
|                            |                                      |  | Register 0 | Register 1   |
| DISPTS (10)                | TS Dispatcher (D)                    | A task switch has taken place resulting in a new time-sharing task being dispatched. | TJID       |  |
| DISPWAIT (11)              | TS Dispatcher (D)                    | A task switch has taken place resulting in a system Wait.                            |            |  |
| QSCEST (12)                | Region Control Task (S)              | Quiesce is started   | TJID       |  |
| QSCECMP (13)               | Region Control Task (S)              | Quiesce is complete  | TJID       | Number of FBQEs  |
| SWOUTST (14)               | Time Sharing Control Task (SWAP) (S) | Swap out Start   | TJID       |  |
| SWOUTCMP (15)              | Time Sharing Control Task (SWAP) (S) | Swap out complete  | TJID       |  |
| SWINST (16)                | Time Sharing Control Task (SWAP) (S) | Swap in started  | TJID       |  |
| SWINCMP (17)               | Time Sharing Control Task (SWAP) (S) | Swap in complete   | TJID       |  |
| RSTORST (18)               | Region Control Task (S)              | Restore is started   | TJID       |  |
| RSTORCMP (19)              | Region Control Task (S)              | Restore is complete  | TJID       |  |
| (20)                       | Reserved                             |  |            |  |
| (21)                       | Reserved                             |  |            |  |
| (22)                       | Reserved                             |  |            |  |
| (23)                       | Reserved                             |  |            |  |
| (24)                       | Reserved                             |  |            |  |
| LOGACCT (25)               | LOGON (S)                            | Pass Logon information for accounting purposes                                       |            | Address of accounting information  |
| CHGTOD (26)                | Timer SLIH (S)                       | Time of day must be altered  | 0          | TOD change in 52 M value sec<br>Timer units<br><br>Bit 0<br>0 - positive value<br>1 - negative value |

Entry Code Table (Part 3 of 4)

| Event Name<br>(Entry Code) | Calling Routine (CLASS)                             | Reason for Entry                                       | Input      |  |
|----------------------------|---|--|------------|--|
|                            |   |  | Register 0 | Register 1   |
| SPRGNSZ (27)               | Timer Sharing Control Task, Region Control Task (S) | Specify size region for specific region                | 0          | Region number  |
| (28)                       | Reserved  |  |            | Required region size   |
| LOGOFF (29)                | End of Task (S)                                     | TJID is to be released.<br><br>Region can be released. | TJID       |  |
| LOGON (30)                 | LOGON (S)   | Hook user into selected region                         | TJID       | Region ID  |
| REQRGNID (31)              | Time Sharing Control Task, Logon (S)                | Obtain region ID appropriate to size                   | TJID       | Region size  |
| SWINERR (32)               | RCT and Time Sharing Control Task (SWAP) (S)        | Swap in failed   | TJID       | Bit 0<br><br>0: LOGON image<br><br>1: Not LOGON image  |
| SWOTERR (33)               | Time Sharing Control Task (SWAP) (S)                | Swap out failed. No room on SWAP data set              | TJID       |  |
| TGETPUT (34)               | TGET, TPUT (T)                                      | TGET was satisfied; TPUT was satisfied                 | TJID       | Bit 0:<br><br>0 - TGET<br><br>1 - TPUT<br><br>For TGET,<br><br>Bit 1:<br><br>0 - all data transferred<br><br>1 - partial transfer<br><br>Bytes 3 and 4: Characters transferred |
| ATTN (35)                  | Terminal Handler (T) Attention                      | Attention (not line delete)                            |            | Sign bit:<br><br>0 - No exit<br><br>1 - Exit   |
| IOERROR (36)               | Terminal Handler HANG UP(T)                         | Permanent I/O Error Terminal disconnected              | TJID       |  |

Entry Code Table (Part 4 of 4)

| Event Name<br>(Entry Code) | Calling Routine (CLASS)                                 | Reason for Entry   | Input      |  |
|----------------------------|---|--|------------|--|
|                            |   |  | Register 0 | Register 1   |
| TERMDSCN (37)              | Terminal Handler<br>LOGOFF (T)                          | Disconnect terminal<br>logically from TSO                | TJID       |  |
| (38)                       | Reserved  |  |            |  |
| (39)                       | Reserved  |  |            |  |
| RGNFAIL (40)               | Time Sharing Control<br>Task Region Control<br>Task (S) | Region failed  |            | Region ID  |
| DONTSWAP (41)              | Transient Area<br>Handler (S)                           | Do not swap out<br>user                                  | TJID       |  |
| OKSWAP (42)                | Transient Area<br>Handler (S)                           | Allow swap out of<br>user                                | TJID       |  |
| UPDATAACC (43)             | LOGOFF (S)  | Update accounting<br>information for<br>user logging off | TJID       |  |
| FEDIAG (44)                | Serviceability (S)                                      | FE diagnostics<br>recorded in TSO<br>TRACE data set      | 0          | Bit 0:<br>0<br>Bits 1-4:<br>Diagnostic<br>Identifier<br>Bits 5-7: n<br>when 2(n+1)<br>equals number<br>of entries<br>Bits 8-31:<br>address of<br>data to be<br>recorded. |
| ENQWAIT (45)               | Enqueue (S)   | User in enqueue<br>WAIT. Swap him out.                   |            |  |

Note: On entry to the Time Sharing Driver, Register 0 contains either:

- 1) 0 - shown in one table as 0.
- 2) A specific TJID - shown in one table as TJID.
- 3) The TJID of the current tasks - shown in the table as blank.

CLASS refers to the TSO Trace Data Set Processor CODES parameter.

## **Appendix D: Terminal Messages Requiring Installation Action**

The following section contains those TSO terminal messages that are generated for the terminal user but that require the operator to perform certain diagnostic measures before calling IBM for programming support. The messages are **not** listed by message number but are listed alphabetically according to message text; the additional messages associated with each initial message are listed alphabetically under each message. The **KEY** indicate at the left of the message denotes the appropriate operator response for the message he responses are listed in numeric order following the list of messages.

| <u>KEY</u> | <u>MESSAGE</u>  |
|------------|---|
|            | DATA SET dsname NOT ALLOCATED, DATA SET NOT ON VOLUME+<br>CATALOG INFORMATION INCORRECT   |
|            | DATA SET dsname NOT ALLOCATED, SYSTEM OR INSTALLATION ERROR+<br>CATALOG ERROR CODE 14<br>CATALOG ERROR CODE 1C<br>CATALOG I/O ERROR |
| 20         | ERROR IN CONCATENATING LIBRARY DATA SETS<br>INPUT   |
| 9          | DADSM ERROR CODE 4704   |
| 10         | DADSM ERROR CODE 4708   |
| 16         | DADSM ERROR CODE 470C   |
| 20         | DADSM ERROR CODE 4710   |
| 20         | DADSM ERROR CODE 4718   |
| 20         | DADSM ERROR CODE 4730   |
| 12         | DADSM ERROR CODE 4734   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 004   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 008   |
| 5          | DYNAMIC ALLOCATION ERROR CODE 104   |
| 5          | DYNAMIC ALLOCATION ERROR CODE 108   |
| 5          | DYNAMIC ALLOCATION ERROR CODE 10C   |
| 6          | DYNAMIC ALLOCATION ERROR CODE 208   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 268   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 304   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 308   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 30C   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 310   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 314   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 318   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 31C   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 320   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 324   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 328   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 338   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 33C   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 340   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 344   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 350   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 358   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 40C   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 408   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 410   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 414   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 418   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 420   |
| 20         | DYNAMIC ALLOCATION ERROR CODE 424   |
| 17         | DYNAMIC ALLOCATION ERROR CODE 504   |
|            | COMMAND SYSTEM ERROR+   |
| 20         | DEVTYPE FAILED xx FOR DDNAME ddname   |
| 20         | TIOT SEARCH FAILED FOR DDNAME ddname  |
| 10         | OBTAIN ERROR CODE 8 FOR DATA SET dsname   |
| 12         | OBTAIN ERROR CODE 12 FOR DATA SET dsname  |
| 16         | BLDL I/O ERROR  |
| 21         | OBTAIN ERROR CODE 12 FOR FORMAT 4 DSCB  |
| 20         | PARSE ERROR CODE xx   |
| 20         | DEFAULT ERROR CODE xx   |
| 20         | SCAN ERROR CODE xx  |
| 20         | DAIR ERROR CODE xx  |
| 20         | PUTLINE ERROR CODE xx   |
| 20         | GETLINE ERROR CODE xx   |
| 20         | PUTGET ERROR CODE xx  |
| 20         | STACK ERROR CODE xx   |

```

6      OUTPUT QUEUE ERROR
20     SVC 98 RETURN CODE xx

      DATA SET dsname NOT ALLOCATED+
8      INVALID UNIT IN USER ATTRIBUTE DATA SET

      USER ATTRIBUTE DATA SET NOT USABLE+
22     CANNOT OPEN DATA SET
16     BLDL I/O ERROR
16     STOW I/O ERROR
14     I/O SYNAD ERROR synadinfo
16     BACKSPACE ERROR 4

      DATA SET dsname NOT USABLE+
16     BLDL FAILED, PERMANENT I/O ERROR IN DIRECTORY
14     I/O SYNAD ERROR synadinfo
10     OBTAIN ERROR CODE 8
21     OBTAIN ERROR CODE 12
22     CANNOT OPEN DATA SET
16     BLDL I/O ERROR
20     XDAP WRITE FAILED IN TEN TRIES
15     OPEN ERROR CODE xxxx
15     ABEND CODE xxx
16     BLDL ERROR CODE xxx

6      JOB QUEUE I/O ERROR

      CAN NOT COPY INTO DATA SET dsname+
14     I/O SYNAD ERROR synadinfo
16     STOW I/O ERROR

      DATA SET dsname CANNOT BE RESOLVED, SYSTEM ERROR+
20     DEFAULT ERROR CODE xx
3      CATALOG ERROR CODE 14
4      CATALOG ERROR CODE 1C

      ERROR WRITING DATA SET dsname, MEMBER AND ALL MEMBERS
      FOLLOWING NOT COPIED+
14     I/O SYNAD ERROR synadinfo

      INPUT DIRECTORY ERROR, CANNOT COPY DATA SET dsname+
14     I/O SYNAD ERROR synadinfo

      MEMBER member CANNOT BE COPIED+
14     I/O SYNAD ERROR synadinfo

      UNABLE TO COMPLETE UPDATE OF OUTPUT DIRECTORY+
16     INPUT DIRECTORY ENTRY xxxx INCONSISTENCIES FOUND

      UTILITY DATA SET
      DATA SET dsname CANNOT BE RESOLVED, SYSTEM ERROR+
20     DAIR ERROR CODE xx
2      LOCATE ERROR CODE 1
4      LOCATE ERROR CODE 24
20     DEFAULT ERROR CODE xx

19     SYSTEM FAILED, ALL USERS TERMINATED

19     SYSTEM FAILURE, PLEASE LOGON AGAIN

      UNABLE TO DELETE DATA SET dsname+
11     SCRATCH ERROR CODE 4
17     SCRATCH ERROR CODE 6
16     STOW ERROR CODE 16

```

```

SYSTEM ERROR+
    UTILITY DATA SET
20    DATA SET dsname NOT UNALLOCATED, DYANMIC ALLOCATION ERROR CODE xx
20    DATA SET dsname NOT UNALLOCATED, CATALOG ERROR CODE xx

function NOT AVAILABLE FOR language+
20    PROGRAM NO LONGER USABLE

FILE SYSPROC NOT USABLE+
22    CANNOT OPEN DATA SET
14    I/O SYNAD ERROR synadinfo
16    FIND ERROR

HELP DATA SET NOT USABLE+
22    CANNOT OPEN DATA SET
16    FIND I/O ERROR
14    I/O SYNAD ERROR synadinfo

CONTROL STATEMENT DATA SET NOT USABLE+
14    I/O SYNAD ERROR
22    OPEN ERROR

                                LIBRARY
20    ERROR IN CONCATENATING, INPUT DATA SETS

HISTORY NOT AVAILABLE+
  2    LOCATE ERROR CODE 4
  4    LOCATE ERROR CODE 24
21    I/O ERROR DURING OBTAIN, CODE 12
11    DATA SET NOT ON VOLUME

MEMBERS NOT AVAILABLE+
16    DIRECTORY STRUCTURE ERROR
16    I/O SYNAD ERROR DURING DIRECTORY SEARCH synadinfo

BROADCAST DATA SET NOT USABLE+
14    I/O SYNAD ERROR
22    CANNOT OPEN DATA SET

22    BROADCAST DATA SET NOT USABLE, CANNOT OPEN DATA SET

BROADCAST DATA SET NOT ALLOCATED, DATA SET NOT ON VOLUME+
  1    CATALOG INFORMATION INCORRECT

14    BROADCAST DATA SET NOT USABLE, I/O SYNAD ERROR

BROADCAST DATA SET NOT ALLOCATED, SYSTEM ERROR+
  2    CATALOG ERROR CODE 4
  3    CATALOG ERROR CODE 14
  4    CATALOG ERROR CODE 1C
20    DYNAMIC ALLOCATION ERROR CODE 004
20    DYNAMIC ALLOCATION ERROR CODE 008
  5    DYNAMIC ALLOCATION ERROR CODE 104
  5    DYNAMIC ALLOCATION ERROR CODE 108
  5    DYNAMIC ALLOCATION ERROR CODE 10C
  6    DYNAMIC ALLOCATION ERROR CODE 208
20    DYNAMIC ALLOCATION ERROR CODE 210
  7    DYNAMIC ALLOCATION ERROR CODE 214
  8    DYNAMIC ALLOCATION ERROR CODE 21C
20    DYNAMIC ALLOCATION ERROR CODE 268
20    DYNAMIC ALLOCATION ERROR CODE 308
20    DYNAMIC ALLOCATION ERROR CODE 30C
20    DYNAMIC ALLOCATION ERROR CODE 310
20    DYNAMIC ALLOCATION ERROR CODE 314
20    DYNAMIC ALLOCATION ERROR CODE 318
20    DYNAMIC ALLOCATION ERROR CODE 31C

```

20 DYNAMIC ALLOCATION ERROR CODE 320  
 20 DYNAMIC ALLOCATION ERROR CODE 324  
 20 DYNAMIC ALLOCATION ERROR CODE 328  
 20 DYNAMIC ALLOCATION ERROR CODE 338  
 20 DYNAMIC ALLOCATION ERROR CODE 33C  
 20 DYNAMIC ALLOCATION ERROR CODE 340  
 20 DYNAMIC ALLOCATION ERROR CODE 344  
 20 DYNAMIC ALLOCATION ERROR CODE 350  
 20 DYNAMIC ALLOCATION ERROR CODE 358  
 20 DYNAMIC ALLOCATION ERROR CODE 408  
 20 DYNAMIC ALLOCATION ERROR CODE 40C  
 20 DYNAMIC ALLOCATION ERROR CODE 410  
 20 DYNAMIC ALLOCATION ERROR CODE 414  
 20 DYNAMIC ALLOCATION ERROR CODE 418  
 20 DYNAMIC ALLOCATION ERROR CODE 420  
 20 DYNAMIC ALLOCATION ERROR CODE 424  
 20 DYNAMIC ALLOCATION ERROR CODE 504  
 9 DYNAMIC ALLOCATION ERROR CODE 4704  
 10 DYNAMIC ALLOCATION ERROR CODE 4708  
 16 DYNAMIC ALLOCATION ERROR CODE 470C  
 20 DYNAMIC ALLOCATION ERROR CODE 4710  
 20 DYNAMIC ALLOCATION ERROR CODE 4714  
 20 DYNAMIC ALLOCATION ERROR CODE 4718  
 20 DYNAMIC ALLOCATION ERROR CODE 4730  
 20 DYNAMIC ALLOCATION ERROR CODE 4734  
 13 DYNAMIC ALLOCATION ERROR CODE 4738

MEMBERS

HISTORY AND MEMBERS

HISTORY NOT AVAILABLE+  
 21 I/O ERROR DURING OBTAIN, CODE 12  
  
 UNABLE TO LIST CATALOG+  
 4 I/O ERROR DURING LOCATE CODE 24  
 2 LOCATE ERROR CODE 4  
  
 DATA SET ATTRIBUTES NOT AVAILABLE+  
 21 OBTAIN ERROR CODE 12  
 10 OBTAIN ERROR CODE 8  
  
 DIRECTORY INFORMATION NOT AVAILABLE+  
 16 I/O ERROR DURING BLDL  
  
 COMPLETE VOLUME LIST NOT AVAILABLE+  
 6 JFCB EXTENSION NOT AVAILABLE  
 2 LOCATE ERROR CODE 4  
 4 LOCATE ERROR CODE 24  
  
 LABEL INFORMATION NOT AVAILABLE+  
 10 OBTAIN ERROR CODE 8  
 21 OBTAIN ERROR CODE 12  
  
 userid LOGGED OFF TSO AT hh: ON month day, year+  
 5 I/O ERROR ON JOB QUEUE  
 20 ALLOCATION UNSUCCESSFUL  
 21 OBTAIN ERROR  
 20 SYSTEM ERROR  
 20 STEPLIB DATA SET COULD NOT BE OPENED  
 20 JOBLIB DATA SET COULD NOT BE OPENED  
 6 NOT ENOUGH JOB QUEUE SPACE TO EXECUTE LOGON  
  
 20 ATTENTION IGNORED, SYSTEM ERROR, LOGON RESUMED  
  
 20 LOGON TERMINATED, SYSTEM ERROR  
  
 20 LOGON TERMINATED, routine ERROR xxx

```

20 LOGON FAILED
20 ABEND WHILE PROCESSING BROADCAST MESSAGES, LOGON PROCEEDING
UNABLE TO RENAME DATA SET dsname+
10 RENAME ERROR CODE 4
10 RENAME ERROR CODE 8
2 CATALOG ERROR CODE 4
3 CATALOG ERROR CODE 16
4 CATALOG ERROR CODE 1C
16 STOW ERROR CODE xx
16 BLDL ERROR CODE xx
10 NOT ENOUGH DIRECT ACCESS SPACE TO CONTAIN ALL RECORDS+
15 SYSTEM ABEND CODE code
UNABLE TO PROTECT DATA SET+
23 I/O ERROR IN PASSWORD DATA SET
UNABLE TO MODIFY PROTECTION FLAGS OF DATA SET dsname+
21 I/O ERROR WHILE UPDATING SECURITY FLAGS
DATA SET RENAMED BUT dsname STILL CATALOGED+
2 CATALOG ERROR CODE 4
3 CATALOG ERROR CODE 16
4 CATALOG ERROR CODE 1C
24 NO SPACE IN DIRECTORY FOR ALIAS
UNABLE TO CATALOG, dsname+
2 CATALOG ERROR CODE 4
3 CATALOG ERROR CODE 16
4 CATALOG ERROR CODE 1C
INPUT DATA SET dsname NOT USABLE+
14 I/O SYNAD ERROR synadinfo
15 INPUT OPEN ERROR CODE xxx
OUTPUT DATA SET FOR JOB jobname NOT USABLE+
14 SYNAD ERROR synadinfo
15 OPEN ERROR CODE xxx
10 NOT ENOUGH DIRECT ACCESS SPACE
UNABLE TO QUALIFY dsname+
2 DEFAULT ERROR CODE xx LOCATE CODE 4
4 LOCATE CODE 24
20 DEFAULT ERROR CODE xx
PRINT DATA SET NOT USABLE+
20 PERMANENT I/O ERROR
22 CANNOT OPEN DATA SET
SYMBOL ADDRESS NOT AVAILABLE, SYSTEM ERROR+
14 I/O SYNAD ERROR synadinfo
22 CANNOT OPEN DATA SET, DDNAME dname
command name ENDED DUE TO AN ERROR
SUBCOMMAND NAME
19 USER ABEND CODE IS code
19 COMPLETION CODE IS code
19 SYSTEM ABEND CODE IS code
19 LINK TO SUBCOMMAND FAILED
19 INSUFFICIENT STORAGE
19 SYSTEM CODE code INSUFFICIENT STORAGE
19 SYSTEM CODE code PERMANENT ERROR DURING BLDL
19 SYSTEM CODE code GETMAIN FAILURE
19 SYSTEM CODE code LINK FAILURE

```

```

19      SYSTEM CODE code ATTACH FAILURE
19      SYSTEM CODE code

      subcomm FAILED+
          ABEND CODE SYSTEM = code
          ABEND CODE SYSTEM = code IC = addr
          INSTR IMAGE = image
20      LINK TO PARSE FAILED
20      LOAD FAILED
20      ATTACH FAILED
20      OPEN FAILED
20      LINK TO SYM FAILED
20      LINK TO DAIR FAILED
20      LINK TO SCAN FAILED
20      XCTL FAILED
20      LINK TO DEFAULT FAILED
20      GETMAIN ERROR CODE xxx

      subcommand FAILED, COMMAND SYSTEM ERROR+
20      PARSE ERROR CODE xx
20      DAIR ERROR CODE xx
20      GETLINE ERROR CODE xx
20      PUTLINE ERROR CODE xx
20      PUTGET ERROR CODE xx

      UNABLE TO LOAD PROGRAM+
22      OPEN ERROR
16      BLDL ERROR CODE xx

```

#### Key Explanation

1. If the data set was deleted by a utility program that was also supposed to update the catalog, but the catalog was not updated, do the following before calling IBM for programming support:
  - Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IEHLIST system utility program to obtain of the catalog and a listing of the volume table of contents (specifying the FORMAT option) of the associated volume, and save the resulting output.
  - Execute the IEHPROGRM system utility program to uncatalog the data set.
2. If the necessary control volume is mounted and the error persists, do the following before calling IBM for programming support:
  - Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IEHLIST system utility program to obtain a printed copy of the catalog, and save the resulting output.

4. Do the following before calling IBM for programming support:
  - Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IEHLIST system utility program to obtain a printed copy of the catalog, and save the resulting output.
  
5. Do the following before calling IBM for programming support:
  - Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IMCJQDMP system utility program to obtain a printed copy of the job queue, and save the resulting output. IMCJQDMP - specify JOBNAME or TOTAL option.
  
6. If sufficient space exists in the SYS1.SYSJOBQE data set, do the following before calling IBM for programming support:
  - Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IMCJQDMP service aid program to obtain a formatted copy of the contents of the SYS1.SYSJOBQE data set, and save the resulting output, specifying the total option.
  - Execute the IMDSADMP service aid program, specifying the TYPE=HI option, to write the contents of main storage to a tape volume. After restarting the system, execute the IDMRDMP service aid program, specifying the GO statement, to print a main storage dump for the dump tape produced by the IMDSADMP service aid program. (If a tape is not available, execute the IMDSADMP service aid program, specifying the TYPE=LO option, to directly print a main storage dump.) Save the resulting dump output.
  
7. IF the devices of the type indicated in the User Attribute Data Set for use by this user are actually online, do the following before calling IBM for programming support:
  - Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IMDSADMP service aid program, specifying the TYPE=HI option, to write the contents of main storage to a tape volume. After restarting the system, execute the IMDPRDMP service aid program, specifying the GO statement, to print a main storage dump for the dump tape produced by the IMDSADMP service aid program. (If a tape is not available, execute the IMDSADMP service aid program, specifying the TYPE=LO option, to directly print a main storage dump.) Save the resulting dump output.

- Have the person in the installation authorized to use the ACCOUNT command issue the command with the LIST subcommand to list the attributes of the user having trouble.
8. If the device of the type indicated in the User Attribute Data Set for use by this user were actually included in the system at system generation time, do the following before calling IBM for programming support:
    - Retain the console sheet and instruct the terminal user to retain the terminal listing.
    - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
    - Have the system generation output available from Stage I and Stage II.
    - Have the person in the installation authorized to use the ACCOUNT command issue the command with the LIST subcommand to list the attributes of the user having trouble.
  9. If no uncataloged data set has the same name as the new data set being created by the user, do the following before calling IBM for programming support:
    - Retain the console sheet and instruct the terminal user to retain the terminal listing.
    - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
    - Execute the IEHLIST system utility program to obtain a list of the entire catalog and a list of the volume table of contents (specifying the FORMAT option) of all online direct access volumes, and save the resulting output.
  10. If space exists in the volume table of contents of all online volumes, do the following before calling IBM for programming support:
    - Retain the console sheet and instruct the terminal user to retain the terminal listing.
    - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
    - Execute the IEHLIST system utility program to obtain a list of the volume table of contents (specifying the FORMAT option) of all online direct access volumes.
  11. Do the following before calling IBM for programming support:
    - Retain the console sheet and instruct the terminal user to retain the terminal listing.
    - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
    - Execute the IEHLIST system utility program to obtain a list of the volume table of contents of the associated volume.
  12. If there are no DOS volumes online, do the following before calling IBM for programming support:
    - Retain the console sheet and instruct the terminal user to retain the terminal listing.

- Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IEHLIST system utility program to obtain a list of the volume table of contents (specifying the DUMP option) of all online direct access volumes.
13. If the directory space requested is not larger than the first extent, do the following before calling IBM for programming support:
- Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
14. Do the following before calling IBM for programming support:
- Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IEHLIST system utility program to obtain printed copy of the catalog, and save the resulting output.
15. Do the following before calling IBM for programming support:
- Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Consult the 'System Completion Codes' section of this publication for the associated error code, and respond as indicated after the statement 'If the problem recurs, do the following before calling IBM for programming support' in the Programmer Response for that code; however, do **not** obtain a storage dump and do **not** specify MSGLEVEL=(1,1) in the JOB statement, even if so requested.
16. Do the following before calling IBM for programming support:
- Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IEHLIST system utility program to obtain a printed copy of the directory of the data set.
17. If the required volume is mounted, do the following before calling IBM for programming support:
- Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.

- Execute the IEHLIST system utility program to obtain a list of the catalog, and save the resulting output.
  - Execute the IMDSADMP service aid program, specifying the TYPE=HI option, to write the contents of main storage to a tape volume. After restarting the system, execute the IMDPRDMP service aid program, specifying the GO statement, to print a main storage dump from the dump tape produced by the IMDSADMP service aid program. (If a tape is not available, execute the IMDSADMP service aid program, specifying the TYPE=LO option, to directly print a main storage dump.) Save the resulting dump output.
18. Do the following before calling IBM for programming support:
- Execute the IMDPRDMP service aid program to print the TSO dump data set and the SWAP data sets.
19. Do the following before calling IBM for programming support:
- Inform the user that if a subcommand was being processed, any further information must be obtained from the user; that is, the user should:
    - 1) Use the ALLOCATE command to allocate a SYSOUT data set with the FILE name of SYSUDUMP.
    - 2) Repeat the error situation.
    - 3) Strike the 'return' key after receiving the message 'xxxxxx ENDED DUE TO ERROR+'
    - 4) Use the FREE command to free the FILE named SYSUDUMP.
    - 5) Retain the terminal listing.
  - Retain the SYSOUT listing containing the dump output.
20. Do the following before calling IBM for programming support:
- Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
21. Do the following before calling IBM for programming support:
- Retain the console sheet and instruct the terminal user to retain the terminal listing.
  - Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
  - Execute the IEHDASDR system utility program to obtain a list of the volume table of contents (VTOC) of the associated volume.
22. If the file is already allocated do the following before calling IBM for programming support:
- Retain the console sheet and instruct the terminal user to retain the terminal listing.

- Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.

If the file is not allocated and you are currently logged on, use the ALLOCATE command to allocate the data set.

23. Do the following before calling IBM for programming support:

- Retain the console sheet and instruct the terminal user to retain the terminal listing.
- Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data sets, the SYS1.SVCLIB data set, and the command library.
- Execute the IEHDASDR system utility program to obtain a printed copy of the Master Password Data Set.

24. If sufficient space exists in the directory do the following before calling IBM for programming support:

- Retain the console sheet and instruct the terminal user to retain the terminal.
- Execute the LISTIDR function of the IBMLIST service aid program to obtain a list of all members with a PTF or local fix, and save the resulting output. Execute the program against the SYS1.LINKLIB data set, the SYS1.SVCLIB data set, and the command library.
- Execute the IEHLIST system utility program to obtain a printed copy of the directory of the data set.

## Appendix E: Glossary

The following terms are defined as they are used in this manual. If you do not find the term you are looking for, refer to the IBM Data Processing Glossary, GC20-1699.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard Vocabulary for Information Processing, which was prepared by Subcommittee X3K5 on Terminology and Glossary of American National Standards Committee X3.

ANSI definitions are preceded by an asterisk. The symbol (SC1) at the beginning of a definition indicates that it has been discussed and agreed upon at meetings of the International Organization for Standardization Technical Committee 97/Subcommittee 1, and has also been approved by ANSI and included in the American National Standard Vocabulary for Information Processing.

**address stop.** In systems with the time sharing option (TSO), a capability to specify at the system console an address which when fetched causes a halt in processing.

**attention exit routine.** In systems with the time sharing option (TSO), a routine that receives control when an attention interruption is received by the system. It allows the user to stop the current command or subcommand and issue a new one.

**attention interruption.** An interruption of instruction execution caused by a remote terminal user hitting the attention key. See also **simulated attention**.

**attention key.** A function key on remote terminals that causes an interruption of execution by the central processing unit.

**attribute.** A characteristic; for example, attributes of data include record length, record format, data set name, associated device type and volume identification, use, and creation date.

**background.** (1) In multiprogramming, the environment in which low-priority programs are executed. (2) In systems with the time sharing option (TSO), the environment in which jobs submitted through the SUBMIT command or SYSIN are executed. One job step at a time is assigned to a region of main storage, and remains in main storage to completion. Contrast with **foreground**.

**background job.** (1) A low-priority job, usually a batched or non-interactive job. (2) In systems with the time sharing option (TSO), a job entered through the SUBMIT command or through SYSIN.

**background reader.** A system task started by the operator to process foreground-initiated background jobs.

**BASIC.** An algebra-like language used for problem solving by engineers, scientists and others who may not be professional programmers.

**batch processing.** (1) \* Pertaining to the technique of executing a set of computer programs such that each is completed before the next program of the set is started. (2) \* Pertaining to the sequential input of computer programs or data. (3) \* Loosely, the execution of computer programs serially. (4) In systems with the time sharing option (TSO), the processing of one job step in a region, so called because jobs are submitted in a group or batch.

**break.** (1) To interrupt the sending end and take control of the circuit at the receiving end. (2) See **receive interruption**.

\* **breakpoint.** A place in a routine specified by an instruction, instruction digit, or other condition, where the routine may be interrupted by external intervention or by a monitor routine.

\* **central processing unit.** (SC1) A unit of a computer that includes the circuits controlling the interpretation and execution of instructions. Synonymous with main frame. Abbreviated CPU.

**Code and Go FORTRAN.** A version of FORTRAN IV for rapid compilation and execution of programs.

**command.** (1) \* A control signal. (2) \* Loosely, an instruction in machine language. (3) \* Loosely, a mathematical or logic operator. (4) In systems with the time sharing option (TSO), a request from a terminal for the execution of a particular program, called a command processor.

\* **command language.** A source language consisting primarily of procedural operators, each capable of invoking a function to be executed.

**command library.** In systems with the time sharing option (TSO), a partitioned data set consisting of command processor programs. A user command library can be concatenated to the system command library.

**command mode.** In systems with the time sharing option (TSO), the entry mode immediately following LOGON, or following completion of a command processor. In command mode, the system is ready to accept any command in the command libraries.

**command procedure.** In systems with the time sharing option (TSO), a data set or a member of a partitioned data set containing TSO commands to be performed sequentially by the EXEC command.

**command processor.** In systems with the time sharing option (TSO), a problem program executed as a result of entering a command at the terminal. Any problem program can be defined as a command processor by assigning a command name to the program and including the program in a command library. Abbreviated CP.

**communication line.** Any medium, such as a wire or a telephone circuit, that connects a remote terminal with a computer.

**component.** (1) In teleprocessing, one or more input/output devices attached to a single control unit, and together making up one remote terminal. (2) A point in a communications system at which data can enter or leave; an input/output device. A component is always attached to a terminal control unit.

**context editing.** In systems with the time sharing option (TSO), a method of editing a line data set without using line numbers. To refer to a particular line, all or part of the contents of that line are specified.

**control terminal.** In systems with the time sharing option (TSO), any active terminal at which the user is authorized to enter commands affecting TSO operation.

**conversational.** Pertaining to a program or a system that carries on a dialog with a remote terminal user, alternately accepting input and then responding to the input quickly enough for the user to maintain his train of thought.

**conversational remote job entry.** An operating system facility for entering job control language statements from a remote terminal, and causing the scheduling and execution of the jobs described. The terminal user is prompted for missing operands or corrections. Abbreviated CRJE.

**CP.** Command processor.

\* **CPU.** Central processing unit.

**CPU time.** The amount of time devoted by the central processing unit to the execution of instructions.

**CRJE.** Conversational remote job entry.

**current line pointer.** In systems with the time sharing option (TSO), a pointer that indicates the line of a line data set with which the user is currently working. A terminal user can refer to the value of the current line pointer by entering an asterisk with EDIT subcommands.

**cursor.** A movable spot of light on the cathode ray tube of a console or display unit that indicates where the next character will be entered.

**DAIR.** Dynamic allocation interface routine.

**data base.** A collection of data fundamental to an enterprise.

**data set extension.** In systems with the time sharing option (TSO), a control block containing control information for each of a terminal user's data sets. Abbreviated DSE.

**DCA. Driver control area.**

**DCARE. Driver control area region extension.**

\* **debug.** To detect, locate, and remove mistakes from a routine or malfunctions from a computer. Synonymous with troubleshoot.

**decay constant.** In systems with the time sharing option (TSO), a weighting factor used in calculating the duration of a job's next time slice based on its use of previous time slices. Recent time slices are more heavily weighted than earlier time slices.

**dedication.** Pertaining to the assignment of a system resource -- an I/O device, a program, or a whole system -- to one application or purpose.

**default value.** The choice among exclusive alternatives made by the system when no explicit choice is specified by the user.

**delimiter.** (1) \* A flag that separates and organizes items of data. Synonymous with separator. (2) A character that groups or separates words or values in a line of input.

**dial-up terminal.** A terminal on a switched line.

**driver control area (DCA).** A control block representing the current state of the time-sharing system, with a section for each time-sharing region; maintained by the driver.

**driver control area region extension (DCARE).** A section of the driver control area representing a time-sharing region.

**driver parameter area.** A parameter list containing information for the time sharing driver.

**DSE. Data set extension.**

**dynamic allocation.** Assignment of system resources to a program at the time program is executed rather than at the time it is loaded into main storage.

**dynamic allocation interface routine.** In systems with the time sharing option (TSO), a service routine that performs various data management functions for command processors. Abbreviated DAIR.

**edit mode.** In systems with the time sharing option (TSO), an entry mode under the EDIT command that accepts successive subcommands suitable for modifying an existing line data set.

**foreground.** (1) In multiprogramming, the environment in which high-priority programs are executed. (2) In systems with the time sharing option (TSO), the environment in which programs are swapped in and out of main storage to allow CPU time to be shared among terminal users. All command processor programs execute in the foreground. Contrast with **background**.

**foreground-initiated background job.** In systems with the time sharing option (TSO), a job submitted from a remote terminal for scheduling and execution in the background.

**foreground initiation.** Under the Disk and Tape Operating Systems (DOS and TOS), the execution of system routines that process operator commands to start a foreground program.

**function key.** A terminal key, such as the attention key, that causes the transmission of a signal not associated with a printable character. Detection of the signal usually causes the system to perform some predefined function for the user.

**GETLINE.** In systems with the time sharing option (TSO), a service routine used by command processors to obtain input.

**input mode.** In systems with the time sharing option (TSO), an entry mode that accepts successive lines of input under the EDIT command for a line data set. The lines are not checked for the presence of subcommands.

**input stack.** In systems with the time sharing option (TSO), a push-down list of sources of input for GETLINE. Possible sources are the terminal or an in-storage list.

**interaction.** In systems with the time sharing option (TSO), a basic unit used to record system activity, consisting of acceptance of a line of terminal input, processing of the line, and a response, if any. An interaction is recorded when a user task starts its wait for a line of terminal input.

**interaction time.** In systems with the time sharing option (TSO), the time between requests for successive lines of terminal input.

**ITF: BASIC.** A simple, algebra-like language designed for ease of use at a terminal.

**ITF: PL/I.** A conversational subset of PL/I designed for ease of use at the terminal.

**job.** (1) \* A specified group of tasks prescribed as a unit of work for a computer. By extension, a job usually includes all necessary computer programs, linkages, files, and instructions to the operating system. (2) In System/360 operating systems, a collection of related problem programs, identified in the input stream by a JOB statement followed by one or more EXEC and DD statements. (3) In systems with the time sharing option (TSO), the processing done on behalf of one user from LOGON to LOGOFF; one terminal session.

**keyword.** (1) One of the significant and informative words in a title or document which describe the content of that document. (2) A symbol that identifies a parameter. (3) A part of a command operand that consists of a specific character string (such as DSNAME=).

**line data set.** In systems with the time sharing option (TSO), a data set with logical records that are printable lines.

**line deletion character.** In systems with the time sharing option (TSO), a terminal character that specifies that all characters are to be deleted from a line of terminal input.

**line number.** In systems with the time sharing option (TSO), a number associated with a line in a line data set, which can be used to refer to the line.

**line number editing.** In systems with the time sharing option (TSO), a mode of operation under the EDIT command in which lines to be modified are referred to by line number.

**linkage editor.** A processing program that prepares the output of language translators for execution. It combines separately produced object or load modules; resolves symbolic cross references among them; replaces, deletes, and adds control sections, and generates overlay structures on request; and produces executable code (a load module) that is ready to be fetched into main storage.

**link library.** A partitioned data set that is used in fetching load modules referred to in execute (EXEC) statements and in ATTACH, LINK, LOAD, and transfer control (XCTL) macro instructions.

**link pack area.** Under the MVT configuration of the control program, an area of main storage containing reenterable routines from system libraries. Their presence in main storage saves loading time when one is needed. Abbreviated LPA.

**link pack area extension.** In systems with the time sharing option (TSO), an extension of the link pack area containing system routines used only when TSO is operating. It is loaded when TSO is started by the operator.

**loader.** (1) A processing program that combines the basic editing and loading functions of linkage editor and program fetch in one job step. It accepts object modules and load modules created by the linkage editor and generates executable code directly in main storage. The loader does not produce load modules for program libraries.

**local system queue area.** In systems with the time sharing option (TSO), a portion of the foreground (swapped) region used for control blocks that are to be swapped out along with a terminal job. Abbreviated LSQA.

**LPA.** Link pack area.

**LSQA.** Local system queue area.

**main storage.** (1) \* The general purpose storage of a computer. Usually, main storage can be accessed directly by the operating registers. (2) All program addressable storage from which instructions may be executed and from which data can be loaded directly into registers.

**main storage region.** Under the MVT configuration of the control program, a subdivision of the dynamic area that is allocated to a job step or a system task.

**major time slice.** In systems with the time sharing option (TSO), the period of time for which a terminal job is swapped into main storage. Contrast with **minor time slice**.

**master scheduler.** A control program routine that responds to operator commands and initiates the requested actions.

**MCP.** Message control program.

**message.** (1) \* An arbitrary amount of information whose beginning and end are defined or implied. (2) In telecommunications, a combination of characters and symbols transmitted from one point to another on a network.

**message control program.** A program that is used to control the sending or reception of messages to or from remote terminals.

**message handler.** In systems with the telecommunications access method (TCAM), a sequence of user-specified macro instructions that examine and process control information in message headers, and perform functions necessary to prepare message segments for forwarding to their destinations. One message handler is required for each line group having unique message handling requirements.

**message switching.** A telecommunications application in which a message received by a central system from one terminal is sent to one or more other terminals.

**minor time slice.** In systems with the time sharing option (TSO), the time during a major time slice in which the tasks associated with a user have the highest priority for execution. Contrast with **major time slice**.

**multiprogramming with a variable number of tasks.** The IBM System/360 Operating System control program that supervises the execution of a variable number of tasks in main storage, and allocates system resources among them. Abbreviated MVT.

**MVT.** Multiprogramming with a variable number of tasks.

**network.** (1) In teleprocessing, a number of communication lines connecting a computer with remote terminals. (2) The interconnection of electrical components.

**non-switched line.** A connection between a remote terminal and a computer that does not have to be established by dialing.

**nucleus.** That portion of a control program that always remains in main storage.

- \* **offline.** Pertaining to equipment or devices not under control of the central processing unit.
- \* **online.** (1) Pertaining to equipment or devices under control of the central processing unit. (2) Pertaining to a user's ability to interact with a computer.

**operand.** (1) \* That which is operated upon. An operand is usually identified by an address part of an instruction. (2) Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor.

**parse.** In systems with time sharing option (TSO), to analyze the operands entered with a command and build up a parameter list for the command processor from the information.

**password.** In systems with the time sharing option (TSO), a one- to eight-character symbol that he may be required to supply at the time he logs on the system. The password is confidential, as opposed to the user identification. Users can also assign passwords to data sets.

**priority.** A rank assigned to a task that determines its precedence in receiving system resources. See also **dispatching priority**, **job priority**, **time sharing priority**.

**problem program.** Any program that is executed when the central processing unit is in the problem state; that is, any program that does not contain privileged instructions. This includes IBM-distributed programs, such as language translators and service programs, as well as programs written by a user.

**processor.** (1) \* In hardware, a data processor. (2) \* In software, a computer program that includes the compiling, assembling, translating, and related functions for a specific programming language. COBOL processor, FORTRAN processor. (3) \* Same as **processing program**.

**prompting.** In systems with the time sharing option (TSO), a function that helps a terminal user by requesting him to supply operands necessary to continue processing.

**RCT.** Region control task.

**read-only.** A type of access to data that allows it to be read but not modified.

**real time.** (1) \* Pertaining to the actual time during which a physical process transpires. (2) \* Pertaining to the performance of a computation during the actual time that the related physical process transpires in order that results of the computation can be used in guiding the physical process. (3) Pertaining to an application in which response to input is fast enough to affect subsequent input, such as a process control system or a computer assisted instruction system.

**receive interruption.** The interruption of a transmission to a terminal by a higher priority transmission from the terminal. Synonymous with break.

**region.** See main storage region, overlay region.

**region control task.** In systems with the time sharing option (TSO), the control program routine that handles quiesce/restore and LOGON/LOGOFF. There is one region control task for each active foreground region. Abbreviated RCT.

**remote job entry.** Submission of JCL statements and data from a remote terminal, causing the jobs described to be scheduled and executed as though encountered in the input stream.

**remote terminal.** An input/output control unit and one or more input/output devices attached to a system through a telecommunications control unit.

**response time.** (1) The time between the submission of a item of work to a computing system and the return of results. (2) In systems with the time sharing option (TSO), the time between the end of a block of user input and the display of the first character of system response at the terminal.

**reverse break.** Same as transmit interruption.

**self-defining delimiter.** Any character appearing in the first position of certain character strings in the TSO command language. A repetition of the character within the string is interpreted as a delimiter.

**separator.** (1) In systems with the time sharing option (TSO), a delimiter used to separate multiple fields in an input line to the system. (2) \* Same as delimiter.

**session.** The period of time during which a user engages in a dialog with a conversational time sharing system; the elapsed time from when a terminal user logs on the system until he logs off the system.

**simulated attention.** In systems with the time sharing option (TSO), a function that allows terminals without attention keys to interrupt processing. The terminal is queried periodically for a specified character string. See also **attention interruption**.

**SMF.** System management facilities.

**STAE (specify task asynchronous exit).** A macro instruction that specifies a routine to receive control in the event of the issuing task's abnormal termination.

**STAI (subtask ABEND intercept).** A keyword of the ATTACH macro instruction that specifies a routine to receive control after the abnormal termination of a subtask.

**swap.** In systems with the time sharing option (TSO), to write an image of a foreground job's main storage region to auxiliary storage, and to read another job's main storage image into the region.

**swap allocation unit.** In systems with the time sharing option (TSO), an arbitrary unit of auxiliary storage space into which a swap data set is divided, and by which it is allocated.

**swap data set.** In systems with the time sharing option (TSO), a data set dedicated to the swapping operation.

**swap data set control block.** In systems with the time sharing option (TSO), a control block describing a swap data set, containing a DCB, a space queue, and device dependent control information.

**swap in.** In systems with the time sharing option (TSO), the process of reading an image of a terminal job's main storage region from a swap data set into the appropriate main storage regions.

**swap out.** In systems with the time sharing option (TSO), the process of writing an image of a terminal job's main storage region from main storage to a swap data set.

**switched line.** A communication line in which the connection between the computer and a remote station is established by dialing. Synonymous with dial line.

**syntax checker.** A program that tests source statements in a programming language for violations of the syntax of that language.

**system management facilities.** An optional control program feature of the IBM System/360 Operating System that provides the means for gathering and recording information that can be used to evaluate system usage. Abbreviated SMF.

**system queue area.** A main storage area reserved for control blocks and tables maintained by the IBM System/360 Operating System control program. Abbreviated SQA.

**system resource.** Any facility of the computing system that may be allocated to a task.

**TCAM.** Telecommunications access method.

**telecommunications access method.** A method used to transfer data between main storage and remote or local terminals. Application programs use either GET and PUT or READ and WRITE macro instructions to request the transfer of data, which is performed by a message control program. The message control program synchronizes the transfer, thus eliminating delays for terminal input/output operations. Abbreviated TCAM.

**telecommunications control unit.** Same as transmission control unit.

**teleprocessing.** The processing of data that is received from or sent to remote locations by way of telecommunication lines.

**terminal.** (1) \* A point in a system or communication network at which data can either enter or leave. (2) Any device capable of sending and receiving information over a communication channel. See also **remote terminal**.

**terminal I/O wait.** The condition of a task that cannot continue processing until a message is received from a terminal; that is, until a TGET is issued.

**terminal job.** In systems with the time sharing option (TSO), a foreground job, a session from LOGON to LOGOFF.

**terminal job identification.** In systems with the time sharing option (TSO), a two-byte identification assigned to each terminal job. Abbreviated TJID.

**terminal monitor program.** In systems with the time sharing option (TSO), a program that accepts and interprets commands from the terminal, and causes the appropriate command processors to be scheduled and executed. Abbreviated TMP.

**terminal user.** In systems with the time sharing option (TSO), anyone with an entry in the user attribute data set; anyone who is eligible to log on.

**TGET.** In systems with the time sharing option (TSO), an I/O macro instruction issued by a problem program to obtain a line of input from the terminal.

**time sharing.** (1) \* Pertaining to the interleaved use of the time of a device. (2) A method of using a computing system that allows a number of users to execute programs concurrently and to interact with the programs during execution.

**time sharing control task.** In systems with the time sharing option (TSO), a system task that handles system initialization, allocation of time-shared regions, swapping, and general control of the time-sharing operation. Abbreviated TSC.

**time sharing driver.** In systems with the time sharing option (TSO), an addition to the dispatcher that determines which task is to execute next.

**time sharing interface area.** In systems with the time sharing option (TSO), a control block used for communication between the driver and the time sharing interface program.

**time sharing interface program.** In systems with the time sharing option (TSO), a program that handles all communication between the MVT configuration of the control program and the time sharing driver.

**time sharing option.** An option of the IBM System/360 Operating System that provides conversational time sharing from remote terminals. Abbreviated TSO.

**time sharing priority.** In systems with the time sharing option (TSO), a ranking within the group of tasks associated with a single user, used to determine their precedence in receiving system resources.

**time slice.** (1) A uniform interval of time on the central processing unit allocated for use in performing a task. Once the interval has expired, CPU time is allocated to another task; thus a task cannot monopolize CPU time beyond a fixed limit. (2) In systems with the time sharing option (TSO), a segment of time allocated to a terminal job. See also **major time slice**, **minor time slice**.

**TJID.** Terminal job identification.

**TMP.** Terminal monitor program.

**transmission code.** A code for sending information over communications lines.

**transmission control unit.** An input/output control unit that addresses messages to and receives messages from a number of remote terminals. Synonymous with **telecommunications control unit**. Abbreviated TCU.

**transmission interruption.** The interruption of a transmission from a terminal by a higher priority transmission to the terminal. Synonymous with **reverse break**.

**trivial response.** In systems with the time sharing option (TSO), a response from the system to a request for processing that should require only one time slice; for example, a syntax check of one FORTRAN statement.

**TSC.** Time sharing control task.

**TS dispatcher.** In systems with the time sharing option (TSO), a section of the time sharing interface program executed as part of the operating system dispatcher. It initiates work requested by the driver.

**TSO.** Time sharing option.

**TSO command language.** The set of commands, subcommands, and operands recognized under the time sharing option (TSO).

**tuning.** In systems with the time sharing option (TSO), the process of adjusting system control variables, such as the percentage of time spent on background execution and the number of foreground users, to make the system divide its resources most efficiently for the workload.

**UADS.** User attribute data set.

**user.** Anyone who requires the services of a computing system. See also **terminal user**.

**user attribute data set.** In systems with the time sharing option (TSO), a partitioned data set with a member for each authorized user of TSO. Each member contains the appropriate passwords, user identifications, account numbers, LOGON procedure names, and user characteristics that define the user profile. Abbreviated UADS.

**USERID.** User identification.

**user identification.** In systems with the time sharing option (TSO), a one-to eight-character symbol identifying each system user. Abbreviated USERID.

**user main storage map.** In systems with the time sharing option (TSO), a map of the allocated storage in a user's region, built by the region control task, and used to determine how much of the region needs to be swapped.

**user profile table.** A table of user attributes kept for each active user, built from information in the LOGON command, the UADS, and the logon procedure.

**verification mode.** In systems with the time sharing option (TSO), a mode of operation under the EDIT command in which all subcommands are acknowledged and any textual changes are displayed as they are made.

## Bibliography

Readers interested in the implementation of the system should be familiar with the information in:

**IBM System/360 Operating System:**

**Introduction**, GC28-6534

**MVT Guide**, GC28-6720.

**Job Control Language Reference**, GC28-6704

**System Generation**, GC28-6554.

**Storage Estimates**, GC28-6551.

**TCAM Programming Guide and Reference Manual**, GC30-2024

The following publications provide information specific to the Time Sharing Option:

**IBM System/360 Operating System: Time Sharing Option Guide to Writing a Terminal Monitor Program or a Command Processor**, GC28-6764 discusses those portions of TSO which communicate directly with the terminal user and can be replaced or modified by an installation.

Three publications provide information for terminal users:

**IBM System/360 Operating System: Time Sharing Option Command Language Reference**, GC28-6732 describes the syntax and facilities of the TSO Command Language.

**IBM System/360 Operating System: Time Sharing Option Terminal User's Guide**, GC28-6763 provides general information about TSO for TSO terminal users.

**IBM System/360 Operating System; Time Sharing Option Terminals**, GC28-6762 describes how to use the terminals supported by TSO.

For information about the internal logic of the Time Sharing Option refer to:

**IBM System/360 Operating System: Time Sharing Option**

**Control Program, Program Logic Manual**, GY27-7199.

**Terminal Monitor Program and Service Routines, Program Logic Manual**, GY28-6770.

For information about the internal logic of a specific command or subcommand, see the appropriate Command Processor Program Logic Manual.

For more detail on specific components or subjects discussed in this publication, the following publications may be of interest.

For system operation and management:

**IBM System/360 Operating System:**

**Operator's Reference**, GC28-6691.

**System Management Facilities**, GC28-6712.

For I/O devices and control units:

**IBM 2701 Data Adapter Unit, Component Description**, GA22-6864.

**IBM Component Description, 2702 Transmission Control**, GA22-6846.

**IBM 2703 Transmission Control, Component Description**, GA27-2703.

**IBM 2741 Communications Terminal**, GA24-3415.

**IBM 1050 System Summary**, GA24-3471.

**IBM Component Description, 2260 Display Station -- 2848 Display Control**, GA27-2700.

# Index

Indexes to systems reference library manuals are consolidated in the publication **IBM System/360 Operating System: Systems Reference Library Master Index**, Order No. GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

Where more than one page reference is given, the major reference is first.

\* as current line pointer  
in data set names

Access methods  
available in foreground 16.65-66  
restrictions on 16  
used with TCAM 66  
ACCOUNT command  
Broadcast data set, effect on 89  
UNIT operand 102  
usage 24  
used in defining a UADS 89  
used in defining TSO unit classes 102  
Account number  
definition 24  
with LOGON 24  
use in UADS structure 24  
ACTIVITY operand of driver parameters 98  
ADDR  
operand of LINEGRP macro instruction 77  
syntax 80  
Address stop  
glossary 140  
ALLOCATE command  
compiler data sets 45  
function 29  
ANS COBOL compiler  
define terminal as file 38  
description 36-39  
example 39  
in terminal environment 39,23  
NOPRINT option 37  
TERM option 37  
Apostrophes  
for data set names 27  
Arrays  
in ITF: BASIC 49  
Assembler(F) 44  
Asterisk  
as current line pointer 28  
in data set names 27  
Attention  
(see also simulated attention interruption, attention interruption)  
glossary 140  
Attention exit routine  
description 26,62  
glossary 140  
Attention interruption  
glossary 140  
Attention key  
for line-delete 25  
glossary 140  
handling 62  
simulating 26

Attributes  
glossary 140  
of the data sets 26-27  
Authorizations  
for users 22,32  
Auxiliary Storage Requirements 117-118  
Available execution time  
in minor time slice 72  
Average queue service time 71  
Average region activity 72  
AVGSERVICE operand of driver parameters 98

Background  
definition 10  
glossary 140  
Background execution percentage  
specifying 74,98  
Background jobs  
glossary 140  
naming 33  
submission for terminal 21,32  
Background programs  
developing from terminal 21  
BACKGROUND operand of driver parameters 97  
Background reader  
glossary 140  
installation exits to SUBMIT command 108  
reader parameters 89  
required data sets 89  
sample cataloged procedure 89  
specifying program 89  
Backspace key  
for character-delete 25  
BASIC  
example 50  
glossary 140  
use 22,49  
Basic Telecommunications Access Method  
restriction 16  
Batch processing  
and time sharing 9  
description 9  
glossary 140  
BRDR (see Background Reader)  
Break  
glossary 140  
Breakpoints  
definition 31  
establishing 62  
glossary 140  
BTAM  
restriction 16  
Buffer Control Parameters  
operands 97  
use 95  
BUFFERS  
operand of buffer control parameters 97  
BUFSIZE  
operand of buffer control parameters 97

CALC command 51  
CALL command  
function 31  
to invoke compilers 45  
CANCEL command 32  
installation exit (see OUTPUT command installation exit)  
CANCEL operator command  
from terminal 32  
use of 32

Catalog  
of data sets 29  
Cataloged procedures (see Writing Cataloged Procedures for TSO)  
Central Processing Unit  
glossary 140  
Character-delete character  
definition 25  
specifying 26  
Checker (see OS PL/I Checkout Compiler)  
Checkpoint/Restart  
restriction 16  
CIB (see Command Input Block)  
CLIST data set 63  
CLOCK  
operand of TSO trace data set processor 108  
COBOL  
see "ANS COBOL"  
COBOL command 37  
CODE  
operand of LINEGRP macro instruction 77  
syntax 79  
Code and Go FORTRAN  
example 54  
glossary 141  
use 52  
CODES  
operand of TSO trace data set processor 108  
Commands  
adding 22  
format 24  
glossary 141  
listed 119-121  
to define work 12  
Command analysis 63  
Command capabilities 20-22  
Command Input Block 84  
Command Language  
glossary 141  
Command library  
glossary 141  
use by Terminal Monitor Program 62  
Command mode 26  
glossary 141  
Command name  
definition 24  
use 24  
Command procedure  
CLIST data set 63  
examples 46,47  
for compilers 47  
function 33  
glossary 141  
handling 63  
nested 47  
when used 33,62  
Command processor  
cancelling 26  
completion 62  
definition 12  
design 64  
glossary 141  
invocation 62  
programming languages for 35  
Communication line  
glossary 141  
Compatibility  
background-foreground 16,22  
with CRJE 16,22  
Component  
glossary 141  
Compute-bound jobs 73  
Concurrent processing 15  
Conditional statements  
example 47  
in command procedures 33  
Context editing 28  
glossary 141  
Control routines 57  
Control terminal  
definition  
glossary 141  
use  
Conversational  
glossary 141  
Conversational Remote Job Entry  
glossary 141  
TSO compatible with  
CONVERT command  
for FORTRAN  
for PL/I  
Copy command 29  
CPU see Central processing unit  
CRJE  
see "Conversational Remote Job Entry"  
CROSSREF  
as operator entered parameter 84  
Current line pointer  
glossary 141  
Cursor  
glossary 141  
CUTOFF  
operand of TSOMH macro instruction 76  
CYCLES  
operand of driver parameters 98  
DAIR see Dynamic Allocation Interface Routine  
Data Base  
glossary 141  
Data Definition statement  
dynamic  
in LOGON procedure  
Data set extension  
glossary 141  
Data entry 27  
Data set management commands 29  
Data sets  
allocation 17  
creating 17  
deleting 29  
line 27  
naming 27  
renaming 29  
retrieving 27  
shared 27  
Data set naming conventions 26  
Data set protection  
commands for 20  
Data set security 20  
Data utilities 29  
DD statements  
dynamic 17  
in LOGON procedure 91  
ddname  
for Broadcast data set 88  
for LOGON cataloged procedures 88  
for swap data sets 88  
for TSO dump 87  
for UADS 87  
Debugging  
glossary 142  
see also "testing programs"  
Decay constants  
glossary 142  
region activity 72  
wait time 72  
DECAYACT  
operand of driver parameters 97  
DECAYWAIT  
operand of driver parameters 97  
Dedication  
glossary 142  
Default values  
glossary 142  
in command procedures 33  
in commands 24  
Defining a UADS using the TSC procedure

Defining terminals as data sets  
for MCP 77

Delimiter  
glossary 142

Descriptive qualifier 26

Diagnostics  
ANS COBOL compiler 37  
Assembler 44  
FORTRAN IV (G1) compiler 42  
FORTRAN syntax checker 45  
ITF: BASIC  
ITF: PL/I  
PL/I Optimizing Compiler 43

DIAL  
operand of LINEGRP macro instruction 77  
syntax 79

Dispatcher 68

DISPLAY operator command  
from terminal 32

Driver (see Time Sharing Driver)

Driver Control Area  
glossary 142

Driver Control Region Extension  
glossary 142

Driver entry codes  
(see also TSO Trace Writer)  
defined 124-128  
syntax 124  
used in measuring system performance 106

Driver Parameters  
operands 98-99  
use 93-94

DSPCH  
operand of TSC parameters 97

DTRACE  
as operand of TSOMCP macro instruction 82  
as operator entered parameter 84

dump  
TSO dump  
ddname for 88  
specifying in TSC start parameters 88

Dynamic allocation  
commands for 29  
DD statements 17  
function 17  
glossary 142  
handling 63

Dynamic Allocation Interface Routine  
glossary 142

Dynamic Area  
main storage requirements 117

EDIT command  
entry modes 28  
for Assembler 44  
for COBOL 36  
for FORTRAN 41  
for PL/I 44  
function 27-28

Editing  
by context 28  
by line number 28

Edit mode  
glossary 142

Entry modes  
definition 26  
for CALC 51  
for EDIT 28  
operator 26

Estimated Wait Time Percentage 73

Even dispatching 73

EXEC command 46,33

EXEC statement  
used to specify background reader 89  
used to specify message control program 86  
used to specify terminal monitor program 91  
used to specify time sharing control task 87  
used to specify TSO trace writer 90

Foreground  
glossary 142

Foreground initiated background job capability 32,21  
glossary 142

Foreground region  
assignment to 70  
definition 10  
main storage requirement 117  
subpools in 117

Format control records  
for TEXT data sets 17

Formatting text 17

Format Command 17

FORT command

FORTTRAN  
choice 22  
list I/O 53

FORTTRAN (E)

FORTTRAN (G1)  
example 42  
options 41  
program entry 41  
testing 42

FORTTRAN (G) compiler 42

FORTTRAN (H) 42

FREE command 27

Free-form source statements  
in Code and Go FORTRAN 53

Function key  
glossary 142

GAM  
restriction on 16

Getline service routine  
glossary 142

Graphics Access Method  
restriction 16

Halt command (See Starting and Stopping TSO)

HELP command 34

HELP information 34,9

Hierarchy support  
restriction on 16

IBMUSER 88

Identification qualifier 26

Identification scheme 24  
examples 24  
for LOGON 24

IESQTCAM (see MCP Start Procedure)

IEFPDSI 88

IKJACCNT 89

Implicit EXEC command  
definition 46  
example 47

Initiator  
called by LOGON 60

INLOCKHI  
operand of buffer control parameters 96

INLOCKLO  
operand of buffer control parameters 96

Input editing  
for terminals 25

Input mode  
glossary 142

Input stack  
glossary 142

Installation exits  
for CANCEL command 110  
for OUTPUT command 110  
for STATUS command 110  
for SUBMIT command 108

Interaction time  
definition 71  
glossary 142  
limit 71

- Interactive programs
  - in COBOL 38
  - in Code and Go FORTRAN 52
  - in ITF: BASIC 49
  - in ITF: PL/I 51
  - in PL/I 43
- Interactive Terminal Facility
  - see "ITF:"
- INTVL
  - operand of LINEGRP macro instruction 77
  - syntax 79
- I/O-bound jobs
- ITF:
  - BASIC 49
  - glossary 143
  - PL/I 45
  - glossary 143
  
- Job glossary 143
- Job Control Language
  - in LOGON procedure 60,91
- Job definition
  - terminal jobs 17
- Job scheduling
  - terminals jobs 17,60
  
- Keyword
  - glossary 143
  
- Language processors
  - ANS COBOL 22
  - Assembler 22
  - Code and Go FORTRAN
  - for program development 22
  - FORTRAN (G1) 22
  - invoked by CALL command
  - ITF: BASIC 22
  - ITF: PL/I 22
  - PL/I (F) 22
  - PL/I Optimizing Compiler 22
- Line, communication
  - multi-drop 80,81
  - non-switched 11
  - switched 11
- Line data set
  - glossary 143
- Line-delete
  - character specifying 34
- Line-delete character
  - definition 25
  - glossary 143
  - specifying 34
- Line group
  - definition 77
- Line numbering
  - for COBOL programs 36
  - for data sets 25
  - glossary 143
- LINEGRP
  - macro instruction operands of 78
  - macro instruction used tailoring an MCP 77-81
- LINENO
  - operand of LINEGRP macro instruction 77
  - syntax 78
- LINK command
- Link library
  - glossary 143
- Link Pack Area 116
  - glossary 143
  - main storage requirement
- Linkage editor
  - glossary 143
- LINKLIB size
- LIST command 29
- List I/O
  - for Code and Go FORTRAN 52
- LISTCAT command 17
  
- LISTDS command 17
- LISSTA macro instruction
  - operands of 81
  - used in tailoring an MCP 76
- LNUNITS
  - as operator entered parameter 83
- Loaderglossary 143
- LOADGO command
  - example 39
  - function 30
- Local System Queue Area 143
- LOGOFF command
  - to end terminal session 34
- LOGON/LOGOFF Scheduler
- LOGON cataloged procedure
  - defining data sets 91
  - naming a terminal monitor program 91
  - parameters on EXEC statement 91
  - restriction on rollout 91
  - sample 92
  - specifying dynamic allocation 99
- LOGON command
  - entry format 24
  - example 24
  - for identification 25,56
- LOGON pre-prompt
  - exit 111
  - exit sample 114
- LOGON procedure
  - identifying 24
  - invoked by LOGON 60
  - job definition 60,56
- LOGON procedure name
  - definition 11
  - in LOGON command 24
- Long precision
  - in ITF: BASIC 49
- LPA
  - operand of TSC parameters 97
  
- Machine requirements 13
- MACLIB size 18
- Main Storage Hierarchy
  - restriction 16
  - glossary 143
- Major time slice
  - calculation of 70
  - definition 70
- MAP
  - operand of TSC parameters 97
- Master Scheduler Region
  - main storage requirement 113
- Matrices
  - in ITF: BASIC 49
- MAXSWAP
  - operand of driver parameters 98
- MAXOCCUPANCY
  - operand of driver parameters 98
- MCP
  - CSECT name 86,82
  - region priority 86
  - start procedure 87
- Message Control Program
  - defining for time sharing 75
  - for time sharing 75
  - glossary 144
  - main storage requirement 116
  - priority 86
  - tailoring 75
- Messages
  - to users 34
- Minimum configuration 13
- Minimum major time slice
  - specifying 94
- Minor time slice
  - calculation of 94
  - definition 73-74
  - glossary 144

**MINSLICE**  
 operand of driver parameters 98,94  
**Modification protection**  
 for data sets 20  
**Modifying data sets** 28  
**MODIFY operator command**  
 from terminal 22  
**Modularity**  
 of control program 55  
**MONITOR operator command**  
 from terminal 22  
**Multidrop line** 13,80-81  
**Multiple region queues** 71  
**Multiprogramming**  
 description 10  
 glossary 143  
**Multistep jobs**  
 restriction in foreground 16  
  
**Naming conventions**  
 example 27  
 for data sets 26  
**Nested command procedures** 47  
**Non-switched line**  
 definition 11  
 glossary 144  
 use with terminal 11  
**NOTIFY= keyword**  
 on JOB statement 32  
**Nucleus**  
 glossary 144  
 main storage requirement 115  
**Null line**  
 after attention 26  
  
**OCCUPANCY**  
 operand of driver parameters 99  
**OLTEST**  
 as operand of TSOMCP macro instruction 82  
 as operator entered parameter 83  
**On-line test procedure (see OLTEST)**  
**Operand substitution**  
 for command procedures 46  
**OPERATOR command** 32  
**Options (see also Starting TSO)**  
 as operand of TSOMCP macro instruction 82  
 use in starting MCP 82  
**OS PL/I Checkout Compiler**  
 as Program Product 23  
 description of 42  
 prompter in 43  
 supported by TSO 43  
**Output**  
 from background jobs 32  
**OUTPUT command** 32  
**OUTPUT command installation exit**  
 command codes 110  
 parameter format 110  
 return codes 110  
**Overview of system** 67  
**OWAITHI**  
 operand of buffer control parameters 97  
**OWAITLO**  
 operand of buffer control parameters 97  
  
**Parallel swapping**  
 definition 15  
 specifying 88  
**Passwords**  
 definition 11  
 for data sets 20  
 glossary 144  
 with LOGON command 11,24  
**PL/I**  
 choice of processors 23  
 for problem-solving 57  
 for programming 43  
  
**PL/I Checker (see OS PL/I Checkout Compiler)**  
**PL/I (F)** 22,43  
**PL/I Optimizing Compiler**  
 options 43  
 program entry 44  
 program execution 44  
**Preemptive scheduling**  
 definition 94  
**PREEMPT**  
 operand of driver parameters 99  
**PRIORITY**  
 operand of driver parameters 99  
**Problem-solving**  
 Code and Go FORTRAN 52  
 comparison of languages 22  
 ITF: BASIC 49  
 ITF: PL/I 51  
**PROC statement**  
 in command procedures 33  
**Procedure name**  
 for LOGON 24  
**PROFILE command** 34  
**Program development**  
 assembler language 44  
 COBOL 36  
 commands for 22  
 FORTRAN 41  
 introduction 21  
 PL/I 43  
 testing 21  
**Program execution**  
 commands for 17  
**Program Products**  
 listed 122  
**Program protection** 20  
**Prompter routine**  
 definition 35  
**Prompting**  
 for input lines 28  
 for operands 24  
 glossary 144  
 user replies to 24  
**PROTECT command** 20  
**Protection**  
 of data sets 20  
 of programs 26  
**Publications, recommended** 148  
**PURGE SVC** 59  
**PUTGET service routine** 63  
**PUTLINE service routine** 63  
  
**Question mark**  
 reply to prompt 34  
**Queue service time**  
 definition 70  
**Quiescing**  
 control of 59  
 definition 59  
  
**Read protection**  
 for data sets 20  
**Reader/Interpreter**  
 called by LOGON 60  
**Record Overflow Feature**  
 required for swapping 14  
**Recovery management** 56  
**Region control task**  
 glossary 145  
**Region operand of EXEC statement**  
 used to specify MCP region size 86  
**REGSIZE**  
 operand of TSC parameters 97  
**Remote job entry**  
 commands for 32  
 glossary 145

- Remote terminals
  - as COBOL files 38
  - definition 11
  - glossary 145
- RENAME command 29
- Required configuration 13
- RESERVBUF
  - operand of buffer control parameters 97
- Restrictions
  - background SVC use 16
  - BTAM 16
  - Checkpoint/restart 16
  - GAM 16
  - Hierarchy support 16
  - Multistep jobs 16
  - Rollout/rollin 16
  - TESTRAN 16
- Rollout/Rollin
  - restriction in foreground 16
- RUN command
  
- Sample cataloged procedure
  - for logon 92
  - for starting an MCP 87
  - for starting background reader 90
  - for starting TSC 88
  - for starting TSO trace writer 90
- Sample MCP cataloged procedures 87
- Sample TSC cataloged procedure 88
- Sample TSO system parameters 101
- SCAN service routine 63
- SEND command 34
- Sequence field
  - in COBOL statements 36
- Serial swapping
  - definition 15
  - specifying 88
- SERVICE
  - operand of driver parameters 100
- Service routines
  - Dynamic Allocation Interface 63
  - GETLINE 63
  - PARSE 63
  - PUTGET 63
  - PUTLINE 63
  - SCAN 63
  - STACK 63
- Shared Language Component
  - see "ITF:"
- Short precision
  - in ITF: BASIC 49
- Simple dispatching 72
- Simulated attention function
  - definition 26
  - glossary 145
  - handling 26
- Size of SUBMIT job queue (see also SUBMIT operand of driver parameters)
- SLACK
  - operand of buffer control parameters 97
- SLC
  - see "ITF:"
- SMF
  - function 18
  - glossary 145
  - operand of TSC parameters 97
- Specifying a Time Sharing Driver 97
- Specifying contents of TSO Link Pack Area 97
- Specify Terminal Attention Exit (STAX)
  - background restriction on 17
- Specify Terminal Control Character (STCC)
  - background restriction on 17
- STACK service routine 63
- STAE macro instruction
  - glossary 145
- STAI keyword
  - glossary 145
- Starting and Stopping TSO
  - as operator entered parameter 88
  - halting the MCP 89
  - starting the MCP 88
  - starting the TSC 89
  - stopping the TSC 89
- STATUS command
  - installation exits for 110
  - use of 32
- STAX macro instruction
  - background restriction on 16
  - use 63
- STCC
  - background restriction on 16
- STOP operator command
  - handling 58
- Storage map 58
- Subcommands
  - format of 24
  - in edit mode 26
- SUBMIT
  - operand of TSC parameters 97
- SUBMIT command
  - installation exit 108
  - buffer format 109
  - module name 108
  - parameter format 109
  - return codes 109
  - use of 32
- SUBQUEUES
  - operand of driver parameters 100
- SVC 93 (TGET/TPUT)
  - restriction on 16
- SVC 54 (STCC)
  - restriction on 16
- SVC 96 (STAX)
  - restriction on 16
  - use 63
- SVCLIB size 118
- Swap Data Set
  - ddname
    - specifying parallel swapping 88
    - specifying serial swapping 88
  - definition 14
  - devices 14
  - glossary 145
  - size 117
- Swap devices
  - allocation unit sizes 118
  - definition 145
- Swap in 145
- Swap out 145
- Swap load
  - calculation of 71
- SWAPLOAD
  - operand of driver parameters 100
- Swapping
  - and major time slice 71
  - and quiesce/restore 59
  - controlling 58
  - definition 10
  - parallel 15
  - separate channels for 15
  - serial 15
- Switched line
  - definition 11
  - glossary 145
  - use with terminal 11
- Symbolic operands
  - in command procedures 46-47,33
- Syntax checking
  - for problem solving languages 22
  - FORTRAN 41
  - glossary 145
  - in input mode 28
  - PL/I 43
- SYSPARM (see also TSO System Parameters)
  - used in TSC start procedure 87
- SYSUADS (see defining a UADS)
- SYSLBC 88

SYSSWAP 88  
 System  
   catalog 29  
   configuration 13  
   security 20  
 System implementation 75  
 System Management Facilities  
   function 18  
   glossary 145  
   specifying 18,97  
 System Queue Area  
   glossary 145  
   main storage requirement 116  
 SYSTSDP  
 SYS1.PROCLIB  
   used in starting TSO tasks 60  
  
 Tab settings  
   for COBOL programs 36  
 Tailoring a message control program  
   job steps involved 75  
   sample MCP specification 85  
   sample job stream 77  
 Task Control Blocks  
   on ready queue 68  
 TCAM  
   see "Telecommunications Access Method"  
 Telecommunications Access Method  
   function 65  
   glossary 146  
   main storage requirement 116  
   terminals supported 13  
   user interface 65  
 Teletypes  
   use with TSO 13  
 TERM  
   operand of LINEGRP macro instruction 77  
   syntax 79  
 TERMAX  
   operand of TSC parameters 97  
 TERMINAL command  
 Terminal conventions  
   discussion of 24  
   entry modes 26  
   input editing 25  
 Terminal I/O  
   service routines 63  
   TCAM function 65  
   user interface for 63  
 Terminal job  
   definition 17  
   glossary 146  
   LOGON procedure for 91  
 Terminal Messages Requiring Installing Action 128  
 Terminal Monitor Program  
   description 61  
   glossary 146  
 Terminal session  
   started by LOGON 24  
 Terminals, remote  
   definition 9  
   description 11  
   execution of batch jobs from 32  
   glossary 146  
 TEST command  
   function 62  
   to invoke a program 62,44  
 Testing programs  
   is assembler language 44  
   in FORTRAN 42  
   in ITF: BASIC 49  
   in ITF: PL/I 51  
   in PL/I 44  
 Test mode  
   definition 45  
   for assembler language 45  
   for FORTRAN 42  
   for ITF: BASIC 49  
   for ITF: PL/I 51  
  
 TEST processor 62  
 TESTRAN  
   restriction on 16  
 Text processing 29  
 TGET/TPUT  
   background restriction 16  
   glossary 146  
   use of Message Control Program 65  
 TIME command  
   handled by Terminal Monitor Program 61  
 Time sharing  
   control task cataloged procedure 87  
   definition 146  
   different than batch 9  
   starting and stopping 88-89  
 Time sharing algorithms  
   and tuning 17  
   definition 17  
   in Driver 17,70-73  
 Time Sharing Control Task  
   description 57  
   glossary 146  
 Time Sharing Control Region  
   main storage requirement 116  
   obtaining 57  
 Time Sharing Driver (see also Driver)  
   description 56  
   glossary 146  
   parameters for 98-100  
 Time Sharing Interface Program  
   description 56  
   glossary 146  
 Time Sharing Link Pack Area  
   main storage requirement 116  
 Time Sharing Option  
   see "TSO"  
 Time slices  
   and tuning 93  
   calculation of 69-74  
   definition 17  
   glossary 146  
   major 70  
   minor 72  
 TIOC (see Buffer Control Parameters) 96  
 TJID  
   operands of TSO trace data set processor 108  
 Transmission Control Units  
   attached to multiplexor 14  
   glossary 146  
   types 14  
 TRANTAB  
   operand of LINEGRP macro instruction 77  
   syntax 79  
 TRACE  
   as operand of TSOMCP macro instruction 82  
   as operator entered parameter 84  
 Trace Data Set Processor  
   listing if FE diagnostic 90  
   sample job stream 90  
 TSC parameters  
   operands 97  
   use 92  
 TSO  
   general description 9  
   glossary 146  
 TSO link pack area  
   LPA operand of TSC parameters 97  
   relation to operating system 67  
   specifying contents 97  
 TSO system parameters  
   buffer control parameters 95  
   driver parameters 93  
   format 96  
   TSC parameters 92-100  
 TSO Trace Writer (see also Trace Data Set Processor)  
   sample job stream 106  
   driver entry codes 124-127  
   output record format 107  
   operands on EXEC statement 107

**TSOMCP macro instruction**  
 used in tailoring an MCP 75  
 syntax 76  
 operands of 76  
**Tuning**  
 and the Driver  
 definition  
 glossary 147  
 Tuning the time sharing driver 102  
**Turnaround**  
 definition 10  
 time for 10

**UNIT operand of ACCOUNT subcommands** 102  
**UNITNO**  
 operand of LINEGRP macro instruction 77  
 syntax 78  
**UNITSIZE**  
 as operand of TSOMCP macro instruction 82  
 default values 82  
 syntax 82  
**Usage statistics**  
 description 18  
**USAS COBOL**  
 see "ANS COBOL"  
**User Attribute Data Set**  
 authorizations in 20  
 definition 20  
 glossary 147  
 modifying  
 size 118  
 use by LOGON 60  
 use by Terminal Monitor Program 61  
**User identification**  
 definition 11  
 glossary 147  
 in LOGON command 11,24  
**User identification qualifier** 26  
**User Main Storage Map**  
 glossary 147  
 use 58  
**User profile**  
 defining 24  
 glossary 147  
 modifying 24  
 use by Terminal Monitor Program 62  
**User verification**  
 by LOGON 24  
**USERCHG**  
 operand of buffer control parameters 97

**WAIT**  
 operand of driver parameters 100  
**Wait time decay constant** 73  
**Weighted dispatching** 94  
**WHEN statement**  
 example 46-47  
 in command procedures 33  
**Writing cataloged procedures for TSO LOGON**  
 start procedure for BRDR 89  
 start procedure for MCP 88  
 start procedure for TSC 88  
 start procedure for TSO trace 90

**XREF**  
 as operand of TSOMCP macro instruction 83  
 as operator entered parameter (see CROSSREF) 83

**1050 Data Communication System**  
 recommended feature 14  
 use with TSO 13  
**2260 Display Station** 13  
**2265 Display Station** 13  
**2301 Drum Storage**  
 as swap device 14  
**2303 Drum Storage**  
 as swap device 14  
**2305 Fixed Head Storage**  
 as swap device 14  
**2314 Direct Access Storage Facility**  
 as swap device 14  
 required features 14  
**2701 Data Adapter Unit** 14  
**2702 Transmission Control**  
 recommended features 14  
 use with TSO 14  
**2703 Transmission Control**  
 recommended features 14  
**2741 Communication Terminal**  
 recommended features 13  
**3330 Disk Storage Facility**  
 as swap device 14

*Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Cut or Fold Along Line

What is your occupation? \_\_\_\_\_

Number of latest Technical Newsletter (if any) concerning this publication: \_\_\_\_\_

Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Cut or Fold Along Line

**Your comments, please . . .**

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold

Fold

First Class  
Permit 81  
Poughkeepsie  
New York

**Business Reply Mail**  
No postage stamp necessary if mailed in the U.S.A.



Postage will be paid by:

International Business Machines Corporation  
Department D58, Building 706-2  
PO Box 390  
Poughkeepsie, New York 12602

Fold

Fold

System/360 OS TSO Guide (S360-20) Printed in U.S.A. GC28-6698-5



**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)



**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]**