

**Burroughs**

**B 5000/B 6000/  
B 7000 Series  
Generalized Message  
Control System**

**(GEMCOS)**

**USER'S/REFERENCE MANUAL**

**PRICED ITEM**

**Burroughs**

**B 5000/B 6000/  
B 7000 Series  
Generalized Message  
Control System**

**(GEMCOS)**

**USER'S/REFERENCE MANUAL**

Copyright © 1982 Burroughs Corporation, Detroit, Michigan 48232

**PRICED ITEM**

The names, places and/or events depicted herein are not intended to correspond to any individual, group or association existing, living or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual living or otherwise, or that of any group or association is purely coincidental and unintentional.

Burroughs believes that the application package described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, can be accepted for any consequences arising out of the use of this material, including loss of profit, indirect, special, or consequential damages. There are no warranties which extend beyond the program specification.

The Customer should exercise care to assure that use of the application package will be in full compliance with laws, rules and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change. Revisions may be issued from time to time to advise of changes and/or additions.

# TABLE OF CONTENTS

Section	Title	Page
	INTRODUCTION . . . . .	xiii
1	THE GEMCOS SYSTEM . . . . .	1-1
	SYSTEM CAPABILITIES . . . . .	1-1
	Transaction Control Language . . . . .	1-1
	Access Control . . . . .	1-1
	Routing . . . . .	1-2
	Formatting . . . . .	1-2
	Paging . . . . .	1-3
	Dynamic Volume Control . . . . .	1-3
	Recovery . . . . .	1-3
	Network Control . . . . .	1-3
	Summary . . . . .	1-4
2	SYSTEM OVERVIEW . . . . .	2-1
	DATA ORGANIZATION . . . . .	2-1
	Files . . . . .	2-1
	Queue Structure . . . . .	2-1
	DATA BASE RELATIONSHIPS . . . . .	2-4
	INFORMATION FLOW . . . . .	2-4
3	TRANSACTION CONTROL LANGUAGE . . . . .	3-1
	SYNTAX CONVENTIONS . . . . .	3-1
	Metalinguistic Symbols . . . . .	3-1
	Metalinguistic Formulas . . . . .	3-2
	CHARACTER SET . . . . .	3-2
	BASIC SYMBOLS . . . . .	3-3
	BASIC COMPONENTS . . . . .	3-4
	Identifiers . . . . .	3-4
	Generalized Identifiers . . . . .	3-4
	Integers . . . . .	3-5
	Strings . . . . .	3-5
	Logical Values . . . . .	3-6
	DECK DESCRIPTION . . . . .	3-6
	CONTROL SECTION . . . . .	3-7
	DIRECTORY . . . . .	3-8
	DIRECTORY OLD . . . . .	3-8
	DUMP . . . . .	3-8
	DUMP OLD . . . . .	3-8
	GENERATE . . . . .	3-9
	IAD . . . . .	3-9
	LIST . . . . .	3-9
	LIST OLD . . . . .	3-9
	LOG . . . . .	3-9
	LOG ERROR . . . . .	3-9
	LOG INPUT . . . . .	3-9
	LOG OUTPUT . . . . .	3-9
	NOFORMATS . . . . .	3-10
	REGENERATE . . . . .	3-10



# TABLE OF CONTENTS (Cont)

Section	Title	Page
	SEQUENCECONTROL Statement . . . . .	3-10
	UPDATEFMT . . . . .	3-12
DEFINE SECTION . . . . .		3-12
DEFINE Invocation . . . . .		3-13
GLOBAL SECTION . . . . .		3-14
ADMINPROGRAM Statement . . . . .		3-16
BATCHIOKEY Statement . . . . .		3-17
CHECKPOINTINTERVAL Statement . . . . .		3-17
CONTINUOUSPROCESSING Statement . . . . .		3-17
CONTROL AUDIT Statement . . . . .		3-18
CONTROLPERMANENT Statement . . . . .		3-18
CONTROLLEDPAGING Statement . . . . .		3-18
FORMATGENERATOR Statement . . . . .		3-19
QUEUERECORDSIZE Statement . . . . .		3-19
LOGOFFMKE Statement . . . . .		3-20
LOGONMKE Statement . . . . .		3-20
MASTERCOMPUTE Statement . . . . .		3-20
MAXBATCHJOBS Statement . . . . .		3-21
MAXNEWACCESSKEYS Statement . . . . .		3-21
MAXNEWFORMATS Statement . . . . .		3-21
MAXNEWMKES Statement . . . . .		3-21
MAXRUNNING Statement . . . . .		3-22
MULTIPLESTAIDENT Statement . . . . .		3-22
MYNAME Statement . . . . .		3-22
PAGINGPERMANENT Statement . . . . .		3-23
QUEUEBLOCKSIZE Statement . . . . .		3-23
RECALLPROGRAM Statement . . . . .		3-23
SERVICEMESSAGE IDENT Statement . . . . .		3-24
SUBSYSTEMS Statement . . . . .		3-24
SYSTEMMONITOR Statement . . . . .		3-25
SYSTEMMONIDENT Statement . . . . .		3-25
SYSTEMNETWORKCONTROL Statement . . . . .		3-25
SYSTEMNETWORKCONTROLIDENT Statement . . . . .		3-26
SYSTEMOBJECTIOKEY Statement . . . . .		3-26
SYSTEMPREFIX Statement . . . . .		3-27
FORMAT AND FUNCTION SECTION . . . . .		3-27
FUNCTION Declarations . . . . .		3-27
FORMAT Declarations . . . . .		3-30
<format declaration> . . . . .		3-33
<format part list> . . . . .		3-34
<special action list> . . . . .		3-34
<local declaration> . . . . .		3-34
<line count statement> . . . . .		3-35
<page begin-end phrase> . . . . .		3-35
<format description> . . . . .		3-35
<REM or variable list> . . . . .		3-35
<editing specifications> . . . . .		3-36

# TABLE OF CONTENTS (Cont)

Section	Title	Page
	<variable segment> . . . . .	3-36
	<fixed segment> . . . . .	3-36
	<editing phrase list> . . . . .	3-36
	<editing string> . . . . .	3-36
	<location specifier> . . . . .	3-37
	<item phrase> . . . . .	3-37
	<case statement> . . . . .	3-39
	<set variable statement> . . . . .	3-39
	<header declaration> . . . . .	3-40
	<break variable statement> . . . . .	3-40
	<end-of-line statement> . . . . .	3-41
	Editing Errors . . . . .	3-42
	Formatting of MCS Error Responses . . . . .	3-48
	Field Sequence . . . . .	3-49
SYSTEM SECTION		3-50
	ACCESSCONTROL Statement . . . . .	3-51
	ADMINKEY Statement . . . . .	3-52
	ARCHIVALAUDIT Statement . . . . .	3-52
	AUTOENABLE Statement . . . . .	3-53
	AUTORECOVERY Statement . . . . .	3-53
	CLEARRESTART Statement . . . . .	3-54
	FLUSHRECOVERY Statement . . . . .	3-54
	HOST Statement . . . . .	3-54
	RECALLKEY Statement . . . . .	3-55
	SUBSYSTEM Statement . . . . .	3-55
PROGRAM SECTION		3-55
	ACCESSCONTROLPROGRAM Statement . . . . .	3-58
	AP300STATUS Statement . . . . .	3-59
	CHARGECODE Statement . . . . .	3-59
	COMMONSIZE Statement . . . . .	3-60
	CONTROLBIT Statement . . . . .	3-60
	CONVERSATIONSIZE Statement . . . . .	3-60
	DECLAREDPRRIORITY Statement . . . . .	3-61
	EDITOR Statement . . . . .	3-61
	FAMILY Statement . . . . .	3-61
	FORMATMAKER Statement . . . . .	3-62
	HOST Statement . . . . .	3-62
	INACTIVETIMEOUT Statement . . . . .	3-63
	INVALIDMESSAGES Statement . . . . .	3-63
	MAXCOPIES Statement . . . . .	3-63
	MAYNOTBEASSIGNED Statement . . . . .	3-64
	MINCOPIES Statement . . . . .	3-64
	MODIFY Statement . . . . .	3-64
	MULTIPLEINPUTS Statement . . . . .	3-65
	PERMANENT Statement . . . . .	3-65
	REMARKS Statement . . . . .	3-66
	RESTARTPROGRAM Statement . . . . .	3-66
	RETURNMESSAGES Statement . . . . .	3-66

# TABLE OF CONTENTS (Cont)

Section	Title	Page
	SERVICE Statement . . . . .	3-67
	SUBSPACES Statement . . . . .	3-67
	SUBSYSTEM Statement . . . . .	3-67
	TESTPROGRAM Statement . . . . .	3-68
	TIMEOUT Statement . . . . .	3-68
	TITLE Statement . . . . .	3-69
	TRANSMISSIONERRORMESSAGES Statement . . . . .	3-69
	USERCODE Statement . . . . .	3-70
INPUTQUEUE SECTION . . . . .		3-70
	ARCHIVALAUDIT Statement . . . . .	3-71
	AUDIT Statement . . . . .	3-72
	AUDITINPUT Statement . . . . .	3-73
	MEMORYLIMIT Statement . . . . .	3-73
	MKE Statement . . . . .	3-73
	NOINPUT Statement . . . . .	3-74
	QUEUEDEPTH Statement . . . . .	3-75
	RECOVERY Statement . . . . .	3-75
	TIMELIMIT Statement . . . . .	3-75
	WAITFORAUDIT Statement . . . . .	3-76
STATION SECTION . . . . .		3-76
	ADDRESS Statement . . . . .	3-80
	ALTERNATE STATIONS Statement . . . . .	3-80
	ASSIGNTOPROGRAM Statement . . . . .	3-81
	BROADCAST Statement . . . . .	3-81
	COMPUTE Statement . . . . .	3-82
	CONTINUOUSLOGON Statement . . . . .	3-82
	CONVERSATIONAL Statement . . . . .	3-83
	DIALIN Statement . . . . .	3-83
	FIXEDLENGTHMKE Statement . . . . .	3-83
	FLUSHOUTPUT Statement . . . . .	3-84
	FORMATTEDBLOCKEDOUTPUT Statement . . . . .	3-84
	FORMSCOMPOSE Statement . . . . .	3-84
	HOST ACCESSKEY Statement . . . . .	3-84
	IDENTIFY Statement . . . . .	3-84
	INDIVIDUALID Statement . . . . .	3-86
	INTERCEPT Statement . . . . .	3-86
	LINE Statement . . . . .	3-86
	LINEANALYZER Statement . . . . .	3-87
	LOGICALACK Statement . . . . .	3-87
	MAYNOTBEINTERCEPTED Statement . . . . .	3-88
	MAXUSERS Statement . . . . .	3-88
	MYUSE Statement . . . . .	3-88
	NODUPPLICATES Statement . . . . .	3-89
	NOLINE Statement . . . . .	3-89
	NOQUEUE Statement . . . . .	3-89
	NOTITLE Statement . . . . .	3-90
	ONEOUTPUTPERBACKUP Statement . . . . .	3-90

# TABLE OF CONTENTS (Cont)

Section	Title	Page
	PORTSIZE Statement . . . . .	3-91
	PORTSTATION Statement . . . . .	3-91
	PROGRAMACK Statement . . . . .	3-93
	REMARKS Statement . . . . .	3-93
	ROUTEHEADER Statement . . . . .	3-93
	SERVICEMESSAGES Statement . . . . .	3-94
	SETMCSREQUESTSET Statement . . . . .	3-94
	SIGNON Statement . . . . .	3-95
	SPECIAL Statement . . . . .	3-95
	STATIONBIT Statement . . . . .	3-95
	STAMASTER Statement . . . . .	3-96
	STASYSTEM Statement . . . . .	3-97
	STATIONHOSTNAME Statement . . . . .	3-97
	STATIONMESSAGEKEY Statement . . . . .	3-97
	STATIONTIMEOUT Statement . . . . .	3-98
	STATIONYOURNAME Statement . . . . .	3-98
	SYSTEMNETWORKCONTROLSTATION Statement . . . . .	3-98
	TEST Statement . . . . .	3-99
	TRANSACTIONMODE Statement . . . . .	3-99
	TYPE Statement . . . . .	3-100
	USERCONTROL Statement . . . . .	3-101
	USERIDMASK Statement . . . . .	3-101
	VALIDUSERS Statement . . . . .	3-101
	VARIABLEMKEPOSITION Statement . . . . .	3-102
	VARIABLEUSERIDPOSITION Statement . . . . .	3-102
	DEVICE SECTION . . . . .	3-103
	FORMATSIN Statement . . . . .	3-104
	FORMATSOUT Statement . . . . .	3-104
	NONFORMATTEDMKES Statement . . . . .	3-105
	STALIST Statement . . . . .	3-105
	AREA STATION SECTION . . . . .	3-105
	AREA STALIST Statement . . . . .	3-106
	OUTPUTROUTING SECTION . . . . .	3-107
	MESSAGE SECTION . . . . .	3-108
	FILE GENERATION . . . . .	3-110
	Option Control Cards . . . . .	3-111
	TCL-Directing Options . . . . .	3-112
4	APPLICATION PROGRAMS . . . . .	4-1
	TRANSACTION PROCESSORS . . . . .	4-1
	User Program Interface . . . . .	4-1
	Process Program Interface . . . . .	4-5
	GETMESSAGE Procedure . . . . .	4-7
	SENDMESSAGE Procedure . . . . .	4-7
	Port Program Interface . . . . .	4-7
	Editor Program Interface . . . . .	4-12
	Service Program Interface . . . . .	4-14
	OBJECT JOB INTERFACE . . . . .	4-17

# TABLE OF CONTENTS (Cont)

Section	Title	Page
	BATCH JOB INTERFACE . . . . .	4-17
	ALTERNATIVE TO THE BATCH INTERFACE . . . . .	4-18
	USER ROUTING PROCEDURE . . . . .	4-20
	CONVERSATIONAL INTERFACE . . . . .	4-22
5	RECOVERY . . . . .	5-1
	NO RECOVERY . . . . .	5-1
	CHECKPOINT RECOVERY . . . . .	5-2
	SYNCHRONIZED RECOVERY . . . . .	5-2
	TCL Syntax . . . . .	5-3
	Recovery-Related Conventions . . . . .	5-4
	Restart TP . . . . .	5-6
	Restart Area . . . . .	5-7
	Program Interface . . . . .	5-8
	WAITFORAUDIT Program Example . . . . .	5-13
	The Recovery Cycle . . . . .	5-17
	Recovery of a Process Program . . . . .	5-20
	Output Message Analysis . . . . .	5-22
	Archival Recovery . . . . .	5-23
	Recovery of a Non-DBMS Transaction . . . . .	5-25
	Synchronized Recovery of Batch Jobs . . . . .	5-25
	Synchronized Recovery of TP-to-TP Transactions . . . . .	5-26
	Transaction Processing System (TPS) Recovery . . . . .	5-27
	TCL Requirements . . . . .	5-29
	TPS Programming Conventions . . . . .	5-29
	Sample TPS Program: . . . . .	5-29
	Summary . . . . .	5-31
6	FORMATTING AND PAGING . . . . .	6-1
	INPUT FORMATTING . . . . .	6-1
	OUTPUT FORMATTING . . . . .	6-2
	FORMS REQUEST . . . . .	6-2
	PAGED FORMATS . . . . .	6-3
	Display Paging . . . . .	6-3
	Update Paging . . . . .	6-4
	The Paging Dialogue . . . . .	6-5
	FORMAT LIBRARIES . . . . .	6-6
	Library Parameters . . . . .	6-8
	Initialization . . . . .	6-9
	Input Formatting . . . . .	6-11
	Output Formatting . . . . .	6-11
	Paged Format Initialization . . . . .	6-11
	Paged Input . . . . .	6-12
	Paged Output . . . . .	6-13
	Message Recall . . . . .	6-13
	Format Library Updating . . . . .	6-14
	Notes . . . . .	6-14
	DEMONSTRATION FILES . . . . .	6-14
7	ACCESS CONTROL . . . . .	7-1
	INTERNAL MCS ACCESS CONTROL MECHANISM . . . . .	7-1

# TABLE OF CONTENTS (Cont)

Section	Title	Page
	USER-SUPPLIED ACCESS CONTROL MODULE . . . . .	7-1
	Station Bits . . . . .	7-3
	Individual Identification . . . . .	7-3
8	NETWORK MANAGEMENT AND CONTROL . . . . .	8-1
	SYSTEM DEFINED INPUT MESSAGE . . . . .	8-2
	ASSIGN INPUT Message . . . . .	8-2
	CHANGE INPUT Message . . . . .	8-3
	CLOSE INPUT Message . . . . .	8-4
	COMPUTE INPUT Message . . . . .	8-4
	\$ INPUT Message . . . . .	8-9
	INTERCEPT INPUT Message . . . . .	8-9
	MODE INPUT Message . . . . .	8-10
	QUIT INPUT Message . . . . .	8-11
	REFRESH OUTPUT Message . . . . .	8-11
	SWITCH INPUT Message . . . . .	8-12
	SYSTEMCONTROLMESSAGES . . . . .	8-12
	Inclusive Identification List . . . . .	8-13
	Specific Identification List . . . . .	8-14
	ADD Request . . . . .	8-16
	ATTACH Request . . . . .	8-16
	CHANGE Request . . . . .	8-17
	CLEAR Request . . . . .	8-23
	DISABLE Request . . . . .	8-23
	Semantics: . . . . .	8-24
	DUMP Request . . . . .	8-25
	ENABLE Request . . . . .	8-25
	INTERROGATE Request . . . . .	8-27
	MOVE Request . . . . .	8-30
	RECOVER Request . . . . .	8-31
	RELEASE Request . . . . .	8-31
	RUN PROGRAM Request . . . . .	8-31
	STATUS Request . . . . .	8-32
	SUBTRACT Request . . . . .	8-36
	SYSTEM Request . . . . .	8-36
	TABLE Request . . . . .	8-38
	UPDATE Request . . . . .	8-41
	WHERE Request . . . . .	8-45
	ZIP Request . . . . .	8-45
	SYSTEM ERROR MESSAGES . . . . .	8-46
	DATA COMMUNICATION ERROR Message . . . . .	8-46
	INPUT MESSAGE PROCESSING ERROR Message . . . . .	8-49
	OUTPUT MESSAGE PROCESSING ERROR Message . . . . .	8-49
	PROCESSING PROGRAM ERROR Message . . . . .	8-50
	System Service Messages . . . . .	8-51
9	CONTINUOUS PROCESSING . . . . .	9-1
	Option Selection . . . . .	9-1
	On-Line Datacom Dumps . . . . .	9-1



# TABLE OF CONTENTS (Cont)

Section	Title	Page
10	AUXILIARY PROGRAMS . . . . .	10-1
	Message Recall . . . . .	10-1
	Station-To-Station Administrative Messages . . . . .	10-5
11	PROGRAM GENERATION AND MAINTENANCE . . . . .	11-1
	Contents of Release Tape . . . . .	11-1
	Product Generation . . . . .	11-3
	Product Maintenance . . . . .	11-5
	Usage and Examples . . . . .	11-7
12	MCS/NDL INTERFACE . . . . .	12-1
13	COMPUTER-TO-COMPUTER COMMUNICATION . . . . .	13-1
	GEMCOS ROUTEHEADER CONVENTIONS . . . . .	13-1
	Variable Field Routeheader . . . . .	13-1
	Fixed Field Routeheader . . . . .	13-2
	Field Definitions . . . . .	13-2
	Return Messages to Program . . . . .	13-3
	Access Control . . . . .	13-3
	Suspension & Resumption . . . . .	13-3
	Formatting . . . . .	13-4
	FILE TRANSFER PROGRAM . . . . .	13-6
	Operation . . . . .	13-6
	COPY Command . . . . .	13-7
	ABORT Command . . . . .	13-8
	WHAT Command . . . . .	13-8
	NEWTC Command . . . . .	13-8
	CHGTC Command . . . . .	13-9
A	COMMON AREA LAYOUT . . . . .	A-1
B	DATACOM SYSTEM DISK FILES . . . . .	B-1
C	THE CONTROL WORDS ARRAY . . . . .	C-1
	RECALL MESSAGE QUEUE . . . . .	C-10
D	HOW TO READ THE MCS STATION MONITOR . . . . .	D-1
	TYPE 1 ACTION . . . . .	D-1
	TYPE 2 ACTION . . . . .	D-1
	TYPE 3 ACTION . . . . .	D-2
	TYPE 4 ACTION . . . . .	D-2
	TYPE 5 ACTION . . . . .	D-3
E	MCS ERROR MESSAGES . . . . .	E-1
	MESSAGES DISPLAYED ON THE SUPERVISORY CONSOLE . . . . .	E-1
	MESSAGES APPEARING AT STATIONS. . . . .	E-2
	MESSAGES TO A SYSTEM NETWORK CONTROL STATION (SPO) . . . . .	E-2
	RECOVERY-RELATED MESSAGES . . . . .	E-5
F	DUMP CONTROL INFORMATION . . . . .	F-1
G	DOCUMENTATION OF TRANSACTION CONTROL LANGUAGE REPORT LISTING . . . . .	G-1

# INTRODUCTION

With the advent of on-line data processing in a multi-task environment (multiple programs running concurrently), programming requirements have greatly broadened in size and complexity. Modern programming must attack a diverse variety of processing concerns to meet the increasing demands of the multi-task environment, including such problem areas as:

1. Formatting data appropriately for each hardware device in the network.
2. Preventing multiple programs from updating the same record simultaneously.
3. Encoding data properly for transmission to or from the network.
4. Ensuring that messages from the network and from programs running in the central system are properly routed to their destination.
5. Maintaining orderly communication between the computer and devices in the network, particularly in environments where several devices share the same line to cut communication costs.
6. Recovering running programs, data bases, and messages or transactions in the event of failure.
7. Preventing unauthorized access to programs and/or data bases.

To relieve the application programmer of these and other major problems common to the on-line, multitask environment, Burroughs has created a series of software modules which together comprise the B 6000/B 7000 Series Generalized Message Control System (GEMCOS). To meet the unique needs of the user community, the system is available in three distinct versions:

1. Basic version:

- a. Message Control System (MCS).
- b. Transaction Control Language (TCL).
- c. Access control.
- d. Message routing.
- e. Checkpoint recovery.
- f. Network control.
- g. Any other features described in this manual and not explicitly included in the Advanced or Total versions.

2. Advanced version:

- a. All of the features of the Basic version.
- b. Message formatting.
- c. Message paging.
- d. Administrative message switching between stations on the network.
- e. Retransmission of output, upon request.

3. Total version:

- a. All of the features of the Basic and Advanced versions.
- b. Dynamic Volume Control.
- c. Synchronized data base/data communications recovery.

This manual describes the Total GEMCOS version. For users of the Basic version or the Advanced version, only the appropriate sections of the manual apply. The user should note that the size of this manual does not indicate an inherent complexity in the use of GEMCOS. On the contrary, a major consideration from the user viewpoint is that GEMCOS is relatively simple to learn, install, and interface with other software. The manual itself provides many examples; it serves as a user's manual, an operator manual, and, in addition, contains maintenance instructions.

This manual is for use with Burroughs GEMCOS program product as designated by the following style identification numbers:

1. Basic Version: B7000 MCB, B6000 MCB, and B5000 MCB.
2. Advanced Version: B7000 MCA, B6000 MCA, and B5000 MCA.
3. Total Version: B7000 MCT, B6000 MCT, and B5000 MCT.

Sections 1 and 2 of the manual present a general overview of the system and its capabilities. Section 3 describes the syntax conventions used to create the Transaction Control Language used by the system and defines the syntactical format requirements for each section within the language. Application program interfacing is discussed in section 4. Section 5 details recovery and restart procedures for the various program types and includes recommendations for selecting appropriate recovery conventions to achieve optimum efficiency. Input/output formatting, forms request, and display and update paging are discussed in section 6. Use of the GEMCOS-supplied Access Control module is described in section 7. Management and control of the system network is discussed in section 8, which presents the syntax specifications for system-defined input messages, System Control messages, System Error messages, and System Service messages. Section 9 provides a complete discussion of the GEMCOS continuous processing capability. The auxiliary programs used to initiate message recall and create administrative message switching are described in section 10. Section 11 summarizes the contents of the GEMCOS-supplied release tape and contains procedures for compiling and maintaining the Message Control System (MCS). A sample request/control set test environment is documented in section 12 to familiarize the user with the MCS/NDL (Network Definition Language) interface. Appendixes A thru G provide information concerning the Control Area Layout, the Control Words array, Data Communications System disk files, dump control, and MCS error messages.

The material in this manual is supplemented by the following documents which pertain to Message Control Systems, Network Controller Interfaces, and Data Management Systems:

<b>Title</b>	<b>Form Number</b>
B 6000/B 7000 Series Generalized Message Control System (GEMCOS) Capabilities Manual	1100211
B 6000/B 7000 Series Generalized Message Control System (GEMCOS) Formatting Guide	1100344
B 6000/B 7000 DMS II Data and Structure Definition Language (DASDL) Reference Manual	5001498
B 6000/B 7000 DMS II Host Language Interface Reference Manual	5001498
B 6000/B 7000 NDL Language Reference Manual	5001522
B 6000/B 7000 Data Communications Functional Description	5000060
B 6000/B 7000 DCALGOL Reference Manual	5011430
B 7000/B 6000 Input/Output Subsystem Information Manual	5001779



# SECTION 1

## THE GEMCOS SYSTEM

A Message Control System (MCS) provides the data-communications user with a viable interface between the Data Communications Processor (DCP) and the application programs that are to process transactions associated with remote terminals. The DCP is pictured as the "heart" of the physical network input/output (I/O). The MCS is envisioned as the brain of the data-communications system, providing the intelligence necessary for objective decision-making regarding the status of messages once they are assembled. The MCS and DCP(s) work together to provide overall control of the data communications system. As a result, a user's application programs can be designed, and even implemented, independent of the network/terminal environment in which they operate.

Burroughs B 6000/B 7000 Series Generalized Message Control System (GEMCOS) is a software package designed to provide users with an MCS that is tailored to meet the specific requirements of a given installation. The MCS is transaction-oriented and provides users with the flexibility to meet a broad range of throughput and processing requirements. GEMCOS anticipates and provides for a dynamic operating environment, allowing reconfiguration of data communications stations, lines and adapters during the running of the system.

### SYSTEM CAPABILITIES

Flexibility and efficiency are the keywords denoting Burroughs B 6000/B 7000 Series GEMCOS. Following is a list of GEMCOS features and capabilities, all of which are described in greater detail in subsequent sections.

#### Transaction Control Language

The Transaction Control Language (TCL) is utilized to provide information such as message routing criteria and MCS option selection. The TCL is also used to describe access-control requirements, message formatting and paging criteria, re-entrant criteria for User programs, and the types of recovery selected for User programs. In addition, the TCL compiler optionally provides the user with a hard-copy record of the data-communications system description.

The language is free-form in structure, using key words to describe both the environment and the requirements of the data communications user. The result of a TCL compilation is the generation of a set of customized tables which are interpreted by the MCS.

Control cards and internal file names which need to be file-equated are the same as Burroughs' ALGOL and COBOL compilers. This adherence to system standards minimizes the learning process involved in using GEMCOS.

#### Access Control

Access Control is optionally available on a station-by-station basis. If Access Control is declared to be in effect for a given station, the legitimate users of that station can be specified. Additionally, a valid sign-on procedure at the station is required to gain access to the system. Specific limitations can be described, determining which transactions are to be allowed for the particular user signing on at the station. A mechanism is also provided which allows a user-written program to participate in the evaluation of sign-ons prior to granting access.



## Routing

Many types of message routing are provided by the MCS. Messages can be routed from: station to station(s); station to Transaction Processor; Transaction Processor to station(s); Transaction Processor to Transaction Processor; Transaction Processor to broadcast lists (areas); and Batch Transaction Processor to Transaction Processor. In instances where a message is to be routed to a station, a chain of alternate stations can be specified in the TCL, to which the message is routed should the station be out of service.

Station-to-station(s) messages are considered to be administrative messages which flow through the system and are audited and delivered to the specified destination stations.

Routing for station-to-Transaction-Processor transactions is normally determined by examination of the message key (defined in the TCL) which is contained within the message. An option also exists for a station to attach itself to a Transaction Processor. When a station is attached to a Transaction Processor, message keys are not expected by the MCS. Transaction Processors can address output to specific stations on a station-by-station basis, or to a group of stations referred to as an "area." A label is associated with a list of stations in the TCL. This list is considered to be an area, and output messages are broadcast to all stations listed for the area. A rotary list can also be associated with an output area, in which case the output message is delivered to that station within the list which has the shortest queue at request time. The area feature allows users to make a final determination of output distribution after programs are written. This means that Transaction Processors can be written without knowledge of the network with which they interact. Association of area labels and lists occurs at the time of TCL generation.

A Batch Transaction Processor is a special type of user-written program which can read a tape, disk, or card file and route transactions to Transaction Processors via the MCS queues. A Transaction Processor processing a transaction in this mode is expected to issue a response back to the Batch Transaction Processor to indicate completion. This mechanism provides for concurrent batch/on-line updating of the same data fields within a data base, while still allowing synchronized data base recovery.

A Transaction Processor can address output to be submitted to another Transaction Processor. The routing may be either by program number of the destination, as specified in the TCL, or by the message key contained in the routed message.

## Formatting

Formatting of messages, independent of Transaction Processors, is provided as an option via a special subset of the TCL. The formatting feature of GEMCOS allows the user to take advantage of various terminal capabilities while, at the same time, providing device-independence at the application program level. A user writing an application is not required to know hardware-control codes or buffer capacity for the various terminals on the network. Rather, the application programmer is only concerned with data strings at the point of MCS interface. A user independently describes the formatting in the TCL by device class to be performed by the MCS on those data strings.

Some of the capabilities provided under formatting include:

1. Forms retrieval.
2. Format modification without compilation of or interruption to application programs.
3. Paging of both input and output, including forms.
4. Numeric field verification.
5. Variable-length fields, with zero/space fill and right/left justification.
6. Variable-sequence fields.

## Paging

Paging occurs when a string of characters constituting a single logical message must be physically segmented because the string exceeds the buffer capacity of the terminal involved. In effect, the MCS acts as an extension to that terminal's buffer capacity. Multiple types of terminals, each having a different buffer capacity, can be utilized for the same multipage transaction. The physical segmentation for each type varies, based upon buffer capacity. This variation is transparent to the application program involved. Three types of paging are provided: input paging, display paging, and update paging; each of which is described in detail in section 6.

## Dynamic Volume Control

Parallel processing of a wide variety of transactions is considered to be the normal mode throughout the total system. All stations are serviced for output concurrently, and it is not necessary for a message to one station to be delivered before a message to some other station is initiated.

Parallel processing of multiple transactions by a single application program is readily available through the Dynamic Volume Control feature of GEMCOS and can be totally transparent to the application programmer. Operating in this mode allows multiple executions of the same program code to be in process for separate transactions at any point in time. In addition, these multiple executions of program code can run concurrently on multiple-processor systems. Only one copy of the machine code is present in the Dynamic Volume Control mode. This feature enables the user to adapt to changing network demands. Thus, response time can be minimized through parallel processing. Based upon user specifications, the MCS initiates new executions of User programs as dictated by run-time transaction loads. Conversely, as volumes drop, the MCS initiates termination procedures for programs.

## Recovery

GEMCOS provides numerous recovery capabilities within the TCL. The user has the flexibility to analyze application-oriented needs and select the recovery options required on a program-by-program basis.

Recovery capabilities range from a rather conventional checkpoint/restart technique to an automatic-transaction queue, application-programs, and data base rollback-and-synchronization scheme. Once again, the emphasis is on providing users with the flexibility to easily adapt in order to effectively meet a broad range of on-line data processing requirements.

## Network Control

In order to dynamically effect certain levels of control over the data-communications processing environment, the user has the capability to identify certain Network Control stations. Such stations may participate in various network-management type transactions. The MCS notifies these stations of various exceptional conditions as they occur. Network Control stations are permitted to make various inquiries of the MCS. In addition, these stations may dynamically alter a subset of the TCL features. Included within the scope of a Network Control station is the ability to notify the MCS that the on-line network is being physically reconfigured. The system takes appropriate action such that only those stations experiencing reconfiguration need be temporarily out of service.

The flexibility, convenience and efficiency of GEMCOS are further enhanced by the following built-in features:

1. A variety of statistical information concerning stations, programs, and MCS. In addition to real-time accessible statistics, a set of files is provided containing input and output messages. From these files, the user can obtain statistics concerning peak-load conditions, network utilization, response times, etc. This allows fine tuning of a system following initial implementation.
2. Retransmission of output upon request. This feature can be useful in the event of paper jams or tears at a printer.
3. Detection and diagnosis of a variety of error conditions and recovery from such conditions.
4. A Line Analyzer Software module contained within the MCS. This module is designed to work in conjunction with a hardware component (known as the Line Analyzer) to troubleshoot various modem/adaptor-oriented problems.
5. Network Control Station monitoring of transmissions to/from stations on the network, transparent to the network.
6. The capability to pass to Transaction Processors fixed data relating to input stations and/or the particular transaction type.
7. Permanent or Temporary Transaction Processor descriptions. This description determines whether a program normally terminates when there are no input transactions queued for some user-defined period of time. Describing a program as permanent does not mean that the code segments for that program will permanently reside in main memory.
8. An interface between generalized, user-written routines and Transaction Processors which allows the user to perform special functions such as editing, accounting, or file handling procedures, which might be common to all transactions and applications.
9. Administrative message switching between stations on the network.
10. Routing headers for computer-to-computer transmission.
11. Higher-level programming languages. Assembler code is not used on the B 6000/B 7000 Series. The MCS is capable of interfacing combinations of COBOL and ALGOL. A Port type program may be written in PL/I.

## Summary

GEMCOS is a Program Product Development Aid that provides B B 6000/B 7000 Series users with the capability to easily and rapidly obtain a tailored MCS. The package attacks a wide range of data-communications problems that are commonly encountered within the user community. The goal of GEMCOS is to make many such problems transparent to the applications programmer.

Network characteristics, including terminal formats and names, can be transparent to User programs. Dynamic Volume Control provides the flexibility for the running system to adapt, without human intervention, in order to deal effectively with changes in transaction mixture. GEMCOS plays the major role in effectively recovering from failures and in maintaining data base integrity in the process. Further, the user is not locked into attempting to maintain a fixed environment. As application-oriented requirements change, or new hardware needs arise, the user can readily adapt in terms of altering:

1. Central site hardware configuration.
2. Central site software structure.
3. Network configuration, including types of terminals.
4. Application/data base design.

## **SECTION 2**

# **SYSTEM OVERVIEW**

This section outlines the data organization, queue structure, and information flow through GEMCOS. The user need not be familiar with the information contained in this section to set up the communications network. If interested in the structure and information flow, the GEMCO user should refer to figure 2-1.

## **DATA ORGANIZATION**

The data is organized in files and queues, as discussed below.

### **Files**

The primary files maintained by the system are the Input Queue file, the Output Queue file, and the Control file. They are maintained on disk and serve several functions. The Input Queue file serves as the queue for incoming messages and the input audit. The Output Queue file is the vehicle used to queue outgoing messages, audit output data, and retrieve messages for retransmission. The Control file contains pointers into the Input and Output Queue files for recovery purposes. File layouts and descriptions are contained in appendix B.

An additional file, the Format file, is created if the user specifies any formats. It contains coded format descriptions, including information required for message mapping and paging. A file layout and description appear in appendix B.

### **Queue Structure**

Whenever possible, the individual queues maintained within the MCS are discussed in the sequence in which they are utilized. All queues are first-in, first-out (FIFO).

The Primary queue is the interface between the MCS and the DCP. All input is received from the DCP via this queue with the exception of paged messages.

The Current queue is assigned to individual stations in place of the Primary queue when a message is to be paged. Subsequent input from the station passes through the Current queue until the paged message is complete. The station is then re-assigned to the Primary queue.

There is at least one, and at most three, Input queues for each Transaction Processor (TP) plus a system Error Message Input queue. Once an input message is validated by the Input stack, the routing module determines the destination and places the message into the appropriate Input queue.

The Editor queue, consisting of one queue for each row in the Common array, is the recipient of the message at the top of the Input queue. From the Editor queue, the message passes to the Editor program, if one exists, and then to the Common area, for the User program to process. If no Editor program exists, the message passes directly from the Editor queue to the Common array currently used by that User program.

The Output queue is a single queue for output messages to all stations. There is never more than one message per station resident in the Output queue. If more than one output message is ready for a station, the remaining messages for that station are "tanked" in the Output Queue file. As soon as a station acknowledges receipt of a message, the MCS retrieves the next one queued for the station and initiates its transmission.

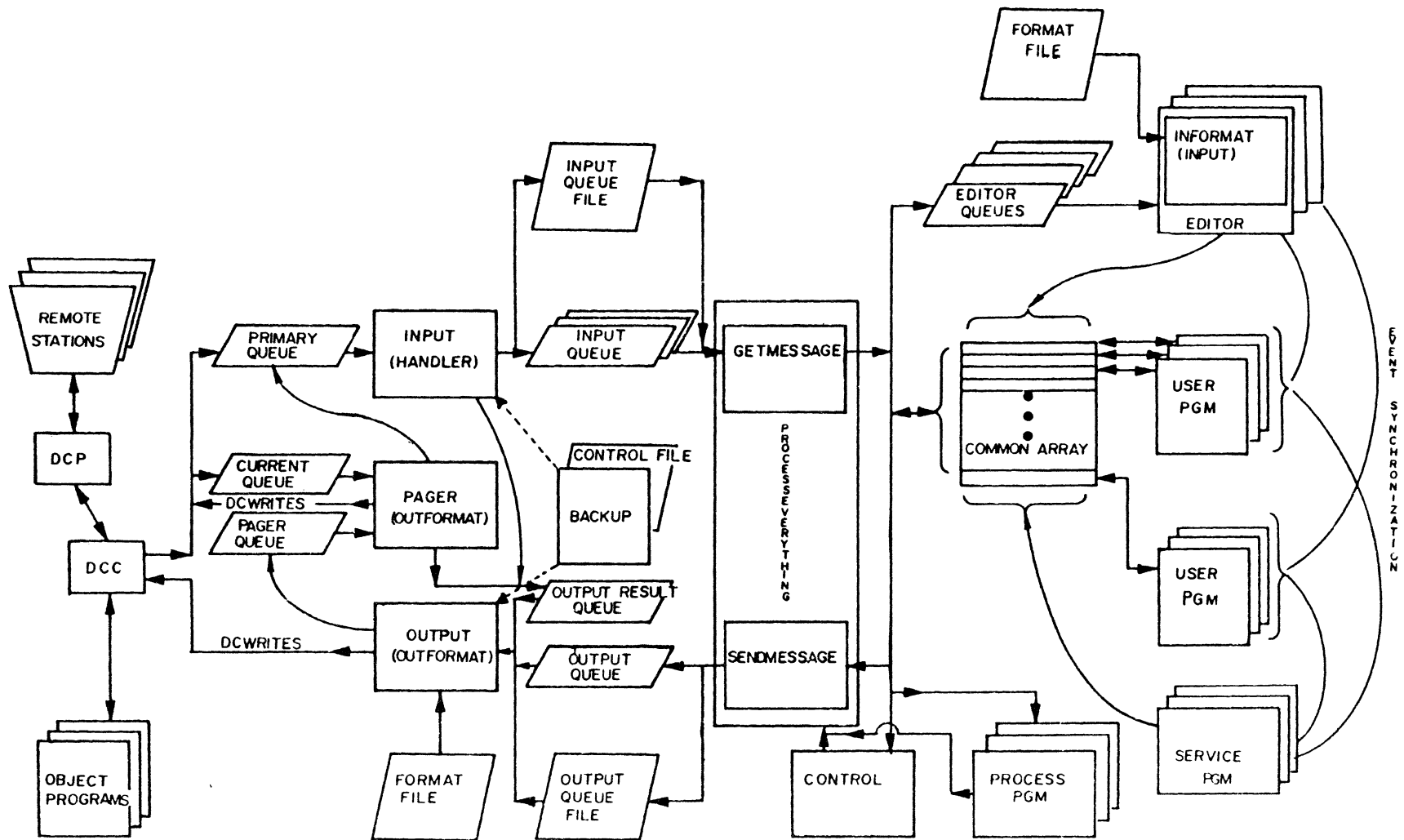


Figure 2-1. 3 7000/B 6000 Series GEMCOS Flow



## DATA BASE RELATIONSHIPS

The MCS manages its own files and does not make use of an external Data Base Management System.

Although the MCS has no direct relationship with an external Data Base Manager, it interacts with, and performs services for, Transaction programs which utilize the services of DMS II. This results in an implicit relationship between the MCS and DMS II, which directly affects some MCS logic. The MCS passes to Transaction Processors (TPs) data which must be stored in restart data sets. The structure of the internal queue files is affected by this relationship as well. The MCS stores data gathered by TPs in its Output Queue file so it can have knowledge of the data base during recovery processing. The details of recovery processing are discussed in section 5.

A standard Restart Transaction Processor is provided. Users wishing to utilize a Restart TP are required to store information in the DMS II Restart Area in a predefined manner. These recovery conventions are specified in section 5.

The MCS is also designed to function effectively in a non-DMS II environment.

## INFORMATION FLOW

The MCS is comprised of three permanent stacks (Input, Output, and Process-everything) which direct and control the information flow within it, and a fourth permanent stack (BACKUP) responsible for checkpointing transient table information. A fifth stack (Control), which is optionally permanent, handles Network Control commands and also plays a role in certain types of message routing.

The Input stack is the main MCS stack. It is the Input stack's responsibility to route messages from the Primary queue to other parts of the system to be acted upon, to audit the input messages as required, and to maintain table information reflecting the current status of the physical and logical characteristics of the network.

All messages in the Primary queue are removed into a DCALGOL message variable (MSG) which involves no movement of data. Upon removal, each message is examined and acted upon according to its type. There are four major types of messages:

1. Results from calls on the DCWRITE function.
2. Messages indicating station or line errors.
3. Messages indicating the results of output transmissions to the network from the Output stack.
4. Input messages from either the stations in the network or from an Object program which has written to a remote file.

Results from calls on the DCWRITE function are handled by updating the relevant in-core tables and causing the event on which the Backup stack "sleeps," if the need exists for the newly updated table(s) to be checkpointed.

Messages indicating station or line errors are linked into the System Error Messages Input queue. The Process-everything stack services this Input queue and generates appropriate error messages which are sent to all System Monitor stations.

Messages indicating the completion of an output transmission are inserted into the Output Result queue in order to trigger the Output stack to send the next output, if any, to the station involved.

Input messages are queued on disk in the MCS Input queue file, which also serves as an audit file. The Input Queue file consists of a series of linked lists, one for each set of transaction codes (Input queues) defined in the TCL, and two system-defined linked lists for System Control messages and System Error messages (Control queue and Error queue) which are handled by the Control and Process-everything stacks, respectively. Each input message is first examined to determine if it is a message to be linked into the System Control Messages' Input queue. This is done by isolating the first syntactic item from the front of the message (or at the message key position, if specified in the TCL for this station) and searching a table of reserved system-defined message keys, among which are the Object job I/O keys. If the message is not a System Control message and the originating station is not ASSIGNED to a TP, a search of a table of transaction codes is made to determine into which Input queue the message should be linked. If no matching transaction code is found in the table, the message is linked into the System Error messages' Input queue to be handled directly by Process-everything, or into a TP's Input queue as specified in the TCL. If a station is ASSIGNED to a TP, the message is placed directly into the TP's Input queue, bypassing the search of the transaction code table.

After determining to which Input queue the message belongs, the message is moved from the MSG variable to a buffer which is then written to the Input Queue file. The MSG is then inserted into its respective in-core processing queue (Iqueue). One Iqueue exists for each linked list within the Input Queue file. An event is then caused, if necessary, to "wake up" the Process-everything stack to further route the message to the correct TP.

The Process-everything stack sleeps on the Master-event (sometimes referred to as the MCS event) which is passed to all application programs. Whenever this event is caused, either by some application program or by some module within the MCS itself, Process-everything wakes up and "looks" for work.

Process-everything first examines all rows of the Common array that are assigned to User programs. If it finds a 1 in Common [0], it knows the User program has indicated there is an output message to be sent to the network. After it picks up and processes output messages, Process-everything looks through all its Input queues. If there is a Transaction Processor ready to receive input, Process-everything passes to it the next queued input message. Having processed all output and input at the application program interface level, Process-everything sleeps, until the Master-event is caused.

The Process-everything stack services TPs in two ways, depending on the type of TP:

1. When servicing a User-type TP, it calls on the Get-message procedure.
2. When servicing a Process-type TP, the Process-everything stack wakes up the TP so that it may call on the Get-message procedure.

(A more detailed description of the various types of application program interfaces is provided in section 4.)

The Get-message procedure is passed a DCALGOL queue as a parameter into which a message from an Iqueue is inserted. In the case of a User-type TP, this queue is the Editor queue associated with its row in Common. If the User TP uses an Editor program, an event associated with the TP's Editor program is caused, waking up the Editor so that the Editor can perform whatever functions are necessary on the message in the Editor's queue. It then moves the "edited" message to the TP's Common area and wakes up the TP. If the TP does not use an Editor, the message is moved directly into the TP's Common area from the Editor queue by Process-everything, and an event is caused which activates the TP.

The discussion up to this point has assumed that the input message can be contained within one input audit file record. In the case of a multiple-record input message, only the last record of the message is inserted into the Iqueue by the Input stack. The Get-message procedure would have to read the additional records from the input file and insert them into its queue parameter together with the last record obtained from the Iqueue. The user is allowed to define the input file's record size in the TCL so that the amount of disk I/Os involved with an input message can be controlled.

After the TP receives an input message in the Common area, it may pass control to an Editor and/or Service program to have various functions performed. As an end result, the TP generates an output message in the Common area and causes the Process-everything event so that the Process-everything stack forwards the output message to the destination designated by the TP in the control words portion of the Common area. The possible types of destinations are a station, a group of stations (Area), all Network Control stations, all System Monitor stations, or another TP. The Process-everything stack calls on the Send-message procedure, which moves the message from a source pointer passed to Send-message as a parameter (which, in the case of a User-type TP, points to the TP's Common area) to an array row corresponding to the TP (Prorecord-area), and from the Prorecord-area to a buffer. The message is then written to the input or output file, depending on the type of destination routing specified by the TP. The Prorecord-area is necessary to hold any intermediate pieces of an output message generated by a TP. (The TP need not pass control to the MCS with a complete output in the Common area but, rather, just a portion of the output message which may not be large enough to write to the input or output file.)

As mentioned previously, Process-type programs are simply awakened by the Process-everything stack as they do their own calls on the Get-message and Send-message procedures which are passed as parameters to all Process-type TPs. The Control stack of the MCS falls into this category and is treated by Process-everything in exactly the same fashion as all other Process-type TPs. The Control stack handles Object job I/O by linking the message into the proper Output queue (a linked list in the MCS' output audit file). In the case of Object job input, i.e., input from a station to an Object job, the Output queue is a reserved linked list in the output file to which all Object job input is routed. Administrative messages (station-to-station or station-to-area) are also handled by the Control stack via the Send-message procedure.

Each time an input message is processed, either by a User or Process TP or the Control stack (or by the Process-everything stack itself in the case of System Error Messages), the Process-everything stack causes the event on which the Backup stack sleeps so that the appropriate in-core Input queue pointer can be checkpointed.

When the Send-message procedure links an output message into a station or Object job's Output Queue file, it checks to see if the Output queue was previously empty. If so, it inserts the output message into the Output stack's Request queue, which wakes up the Output stack if it is sleeping. In the event that at least one or more output messages already exist in the Output Queue file for the station or Object job, this insertion is not done. The Output stack eventually links to this output after sending any previous output messages to the station or Object job. This means that the Output stack must read the Output Queue file to obtain the output message. This convention is used to prevent flooding the Output queue with output messages to any particular station

The Output stack's function is to retrieve messages from the Output queue and/or the Output Queue file into an array. From this array, one block-at-a-time of the message is transposed to a DCALGOL message variable, and a DCWRITE call is made using the message variable. A "block" in this context corresponds to the buffersize of the terminal or Object job to which the message is being forwarded in addition to the physical characteristics (page and width) of the terminal. In the case of unbuffered terminals, a default block size is used. As a result of the DCWRITE call, result messages are generated by the DCP for these blocks. They are received by the Input stack in the Primary queue and are forwarded to the Output stack via the Output Result queue. If the result message indicates a successful operation, the next block of the message is sent, if any blocks remain. If none remain, the next message for the station, if any, is started.

Each time a message is initiated to a station, the first two blocks are immediately sent in order to provide a buffering effect as additional blocks are sent upon receipt of result messages. Upon receipt of the result message for the last block of a message, the Output stack causes the event on which the Backup stack sleeps, so that the station's in-core Output queue pointer can be checkpointed.

As output messages are generated by the Send-message procedure, they are marked as formatted outputs when applicable, with an indication of whether the format to be applied involves multiple pages. The Output and Pager stacks share a global Formatter procedure with which formats are applied to output messages. Nonpaged formatted output is handled directly by the Output stack.

Formatted pages of output are handled by a separate asynchronous stack (Pager) which is invoked by the Output stack. Each output in the Output queue is examined by the Output stack to determine if it is a paged, formatted output; if so, the output is inserted into the Pager queue which the Pager stack services. The Pager stack attaches all stations currently in Paging mode to a single current queue from which all subsequent inputs from these stations are received. Upon completion of the paging process for a station, the Pager stack returns the station to its normal status, and subsequent inputs from the station are received by the Input stack in the Primary queue. At the same time, the Pager stack also routes any updated pages accumulated for the station to the Primary queue.



# SECTION 3

## TRANSACTION CONTROL LANGUAGE

The Transaction Control Language (TCL) provides the user with a means to describe a system in terms of station attributes and routing criteria. In addition, the TCL is used to describe access-control requirements, formatting and mapping criteria, re-entrant criteria for Transaction Processors (TPs), and to provide a variety of other information necessary to the operation of the Message Control System (MCS).

The language has a free-form structure, utilizing key words to describe the data communications environment. As a by-product of a TCL Compilation, the user receives a hard-copy record describing the environment for use as a reference document.

The TCL compiler performs extensive syntax checking of the TCL. In addition, once the driver tables are built, the TCL performs a consistency check.

Syntax errors are textual and are output to the line printer. For user convenience, messages appear immediately after the line in error. Error messages output as the result of the consistency check are also textual, but they appear at the end of the report.

### SYNTAX CONVENTIONS

The following is a discussion of the Backus-Naur form (BNF) used to describe the syntax of the TCL.

#### Metalinguistic Symbols

A metalanguage is a language used to describe other languages. A metalinguistic symbol is a symbol used in a metalanguage to define the syntax of a language. The following metalinguistic symbols are used in this document:

Symbol	Meaning
< >	Left and right broken brackets are used to contain one or more digits and/or letters representing a metalinguistic variable whose definition is given by a metalinguistic formula.
::=	This symbol means: is defined as. The metalinguistic variable to the left of this symbol is defined by the metalinguistic formula on its right.
/	This symbol represents the word OR and is used to separate the various options possible when defining a metalinguistic symbol. Please note that the conventional symbol of a vertical bar is not used in this document to represent the word OR.
[]	Brackets are used to enclose English-language definitions of metalinguistic variables. This formulation is used only when it is impossible or impractical to use a metalinguistic formula.



## Metalinguistic Formulas

Metalinguistic symbols are used in forming a metalinguistic formula. A metalinguistic formula is a rule which produces an allowable sequence of characters and/or symbols. These formulas are used to define the syntax of the TCL. The syntax, combined with the semantics contained in this manual, defines the TCL.

In a metalinguistic formula, any mark or symbol which is not one of those defined above is to be read according to its conventional uses (e.g. periods, commas, etc.). The juxtaposition of the metalinguistic variables and/or symbols in a metalinguistic formula denotes the juxtaposition of those elements in the corresponding construct.

Example:

```
<identifier> ::=  
    <letter> / <identifier> <letter> /  
    <identifier> <digit>
```

This metalinguistic formula is read as follows:

An identifier is defined as a letter, or an identifier followed by a letter, or an identifier followed by a digit.

The metalinguistic formula above defines a recursive relationship by which a construct called an identifier may be formed. That is, evaluation of the formula shows that an identifier begins with a letter. The letter may stand alone, or it may be followed by a sequence of letters and digits.

### NOTE

Beginning with the heading CHARACTER SET, all information contained in this section is presented in the following order: 1) Item to be defined (Set, Description, Section, Statement, Command or Declaration); 2) Syntax; 3) Semantics 4) Pragmatics, if applicable; and 5) Examples, if any.

## CHARACTER SET

The character set for which the language is defined is drawn from the extended binary-coded decimal interchange code (EBCDIC) character set.

Syntax:

```
<character set> ::=  
    <string character> /  
    <string bracket character> / <empty>  
  
<string character> ::=  
    <letter> / <digit> / <special character>
```

<letter> ::=  
     A /B /C /D /E /F /G /H /I /J /K /L /M /  
     N /O /P /Q /R /S /T /U /V /W /X /Y /Z  
  
 <digit> ::= 0 /1 /2 /3 /4 /5 /6 /7 /8 /9  
  
 <special character> ::=  
     . / , / <slash> / = / : / ; / ( / ) /  
     \$ / \* / - / # / @ / & / <single space>  
  
 <slash> ::= /  
  
 <space> ::= <single space> / <space> <single space>  
  
 <single space> ::= [one horizontal position]  
  
 <string bracket character> ::= "  
  
 <empty> ::= [the null string of symbols]

Semantics:

The character set for the MCS and UTILITY define is a 52-character subset of the EBCDIC character set containing letters, digits, special characters, the string bracket character, and the space.

## BASIC SYMBOLS

Syntax:

<basic symbol> ::= <letter> / <digit> / <delimiter>  
  
 <delimiter> ::= <assignment operator> / <separator>  
  
 <assignment operator> ::= =  
  
 <separator> ::= , / . / : / <space> / ; / ( / )

Semantics:

Only upper-case letters are permitted. Delimiters separate various entities that make up a system definition.

## BASIC COMPONENTS

Syntax:

```
<basic component> ::=  
    <identifier> / <generalized identifier> /  
    <integer> / <string> / <logical value>
```

Semantics:

<basic component>s are the primary structures of the language.

### Identifiers

Syntax:

```
<identifier> ::=  
    <letter> / <identifier> <letter> / <identifier> <digit>
```

Semantics:

The maximum length of an identifier is 17 characters. Spaces may not appear as part of an identifier.

### Generalized Identifiers

Syntax:

```
<generalized identifier> ::=  
    <identifier> /  
    <generalized identifier> <slash> <identifier>
```

Semantics:

A <generalized identifier> may contain a maximum of 14 <identifier>s separated by slashes. An <identifier> used as an <identifier component> must be less than or equal to 17 characters.

Example:

COMPUTERROOM/TTY35

## Integers

Syntax:

`<integer> ::= <digit> / <integer> <digit>`

Semantics:

Only positive integers are allowed. A space may not appear within an integer. Integers are limited to eight digits.

## Strings

Syntax:

`<general string> ::=`  
    `<EBCDIC code> <string> /`  
    `<hexadecimal code> <hexadecimal string>`

`<EBCDIC code> ::= 8 / <empty>`

`<string> ::= <EBCDIC string> / <EBCDIC unit string>`

`<EBCDIC string> ::= "<character concatenation>"`

`<character concatenation> ::=`  
    `<string character> /`  
    `<character concatenation> <string character>`

`<EBCDIC unit string> ::=`  
    `"<string character>" / "<string bracket character>"`

`<hexadecimal code> ::= 4`

`<hexadecimal string> ::= "<hex string>"`

`<hex string> ::= <hex pair> / <hex string> <hex pair>`

`<hex pair> ::= <hex character> <hex character>`

`<hex character> ::= 0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F`

`<hex unit string> ::= <hexadecimal code> "<hex pair>"`

```

<1-byte string> ::=
    <EBCDIC code> <EBCDIC unit string> /
    <hexadecimal code> "<hex pair>"

```

Semantics:

The maximum length of a string is 120 characters. Strings containing internal quotes must be broken into separate <strings> containing three quotes in succession.

## Logical Values

Syntax:

```

<logical value> ::= TRUE/FALSE

```

Semantics:

A logical value consists of the two possible conditions that a Boolean may assume.

## DECK DESCRIPTION

Syntax:

```

<deck description> ::=
    <control section> /
    <control section> <define section> <global section>
    <definition section list>;

<definition section list> ::=
    <definition section> /
    <definition section list> <definition section>

<definition section> ::=
    BEGIN
    <system section> <program section> <station section>
    <message section> <area section> <device section>
    <output routing section>
    END

```

Semantics:

The <control section> can be the only section if the tasks requested do not require the <definition section>.

Each <definition section> describes a separate "system" or "application." A system is defined as a group of related stations, programs, areas, devices and messages which are logically separate from other systems. The separation is complete, and inter-system communication is prevented by the MCS. A maximum of eight such "systems" may be declared.

**Example:**

```
CONTROL = LIST, GENERATE;
GLOBAL:
    SYSTEMMONITOR = CMPT.
    SYSTEMSPD = CMPT.
    MAXRUNNING = 3.

BEGIN
    SYSTEM SOS (1):
        PROGRAM EVERYTHING (12) USER:
            INPUTQUEUE IQU1 (1):
                MKE = MKE1 (1,1), MKE2 (1,2), MKE3.
                TITLE = PRG/EVERYTHING.
                COMMONSIZE = 300.
                TIMEOUT = 30.
                REMARKS = "PROCESSES EVERYTHING".
            STATION ONLY/ONE (1):
                LINE = 0:0:2.
                ADDRESS = "AA".
                REMARKS = "THE ONLY STATION".
            MESSAGE TEST = "THE QUICK BROWN FOX JUMPED",
                "OVER THE LAZY DOGS BACK", 4"0D25".
        AREA GENERAL BROADCAST (1):
            STALIST = ONLY/ONE.
    END;
```

## CONTROL SECTION

**Syntax:**

<control section> ::= CONTROL = <control list> ;

<control list> ::=  
    <control task> / <control list>, <control task>

<control task> ::=  
    LIST <list list> / LIST OLD <list list> /  
    <generation task> /  
    DUMP / DUMP OLD / LOG /  
    LOG INPUT / LOG OUTPUT / LOG ERROR /  
    DIRECTORY <directory list> /  
    DIRECTORY OLD <directory list> /  
    UPDATEFMT / NOFORMATS /  
    IAD

<list list> ::=  
    <list task> / <list list> <list task> / <empty>

<list task> ::=  
    STATION / PROGRAM / INPUTQUEUE / AREA / MESSAGE

```

<generation task> ::=
    GENERATE <sequence specification> /
    REGENERATE <sequence specification>

<sequence specification> ::=
    <empty> / ,<sequence control statement>

<directory list> ::=
    <directory task> /
    <directory list> <directory task> /
    <empty>

<directory task> ::=
    PROGRAM / INPUTQUEUE / STATION / LINE / AREA

```

#### Semantics:

The <control section> defines the task that is to be performed.

The <control list> defines the individual task or combination of tasks to be performed.

## DIRECTORY

DIRECTORY produces a report-oriented listing of all programs, Input queues, lines, stations, and areas visible to the system. If a <directory list> is supplied, then a report is generated for only those <directory task>s specified. The current files are used to produce these reports. The format of the Directory listing is presented in appendix G.

## DIRECTORY OLD

DIRECTORY OLD produces a DIRECTORY using the old files. The format of the Directory Old listing is presented in appendix G.

## DUMP

DUMP accepts card or keyboard information and dumps the current files selectively. Syntax and examples for auxiliary programs are presented in appendix F.

## DUMP OLD

DUMP OLD dumps the old files. Syntax and examples for auxiliary programs are presented in appendix F.

## GENERATE

GENERATE creates a new set of files as defined by the <definition section>.

A check is made to determine if there is an existing control file present with the same name as the file to be generated. If this is the case, an error message is written, and no new files are generated. The existing file must either be removed or, if in production mode, have their names changed.

## IAD

IAD indicates that the Control file and/or any other MCS files are present on installation allocated disk. The Utility program does not check the presence of the Control file during a GENERATE or REGENERATE when IAD is specified in the Control statement. Existing MCS files should be preserved since they are rewritten when this syntax is used.

## LIST

LIST causes table information saved on disk to be printed out. If a <list list> is supplied, then table information for only those <list task>s specified is printed. The format of printed table information is presented in appendix G.

## LIST OLD

LIST OLD causes a LIST of the old files to be printed. The internal and external names of files used by the Utility program are presented in appendix B. The format of the printed files is presented in appendix G.

## LOG

LOG creates a log file with one record for each message received and a separate log file with one record for each message sent. The old disk files are used. The format of the log files generated is presented in appendix B.

## LOG ERROR

LOG ERROR lists the messages in the Error Input queue related to data communication problems.

## LOG INPUT

LOG INPUT creates a partial LOG containing only messages received.

## LOG OUTPUT

LOG OUTPUT creates a partial LOG containing only messages sent.



## NOFORMATS

NOFORMATS allows the user to compile the TCL without compiling the formats in the TCL. To avoid format compilation the following actions should be taken.

1. Specify NOFORMATS on the control statement.
2. Place a "\$ SET FORMATS " compiler option statement before the first format or function in the TCL source.
3. Place a "\$ POP FORMATS" compiler option statement after the last format in the TCL source.

The FORMATS compiler option cards will have an effect only if the control card has NOFORMATS specified. The control statement NOFORMATS will have no effect if the compiler option cards are not present.

## REGENERATE

REGENERATE creates a new set of "clean" files as defined in the old control file. All unprocessed input and output messages are copied over from the old set of files into the new set of files. During REGENERATE, the total TCL parameters are not passed or required.

Refer to the GENERATE statement above for the actions taken when a set of files is present with the same name as the files to be generated.

## SEQUENCECONTROL Statement

Syntax:

```
<sequence control statement> ::=  
    SEQUENCECONTROL = <status> <mode> <sequence number>
```

```
<status> ::= FIRST / RERUN / <empty>
```

```
<mode> ::= PRODUCTION / TEST / <empty>
```

```
<sequence number> ::= <integer>
```

Semantics:

Because of the importance of the Data Base Sequence Number (DBSN) in the recovery scheme, this syntax exists to:

1. Carry the current DBSN from an old set of files to a new set of regenerated files.
2. Ensure the continuity of the DBSN by ensuring the sequence of MCS files from one generation to the next.

This is a required statement if any Input queue has been declared with RECOVERY = TRUE.

Two sets of files are referenced: the old set of files which is already in existence (internal names IQUSO, OQUSO, and CQUSO) and the new set which is to be created (internal names IQUS, OQUS, and CQUS). These files may be label-equated.

The <sequence control statement> is used to indicate the SEQUENCE NUMBER of this TCL run. The <sequence number> must be 1 greater than the number used in the old set of files. If the data on this card does not match with the old control file, the TCL terminates without generating a new set of files. Either the proper files must then be loaded or the sequence control card changed to be consistent with the old files. Care must be taken to ensure that the correct old files are being used with the correct sequence control card when creating new TCL files.

The <status> statement indicates the type of TCL run being made. If FIRST is specified, new files are generated without regard to a set of old files. The DBSN is set to 0, and the <sequence number> is inserted into the new files. If RERUN is specified, this is a run of the TCL against a set of old (previously used) files to generate a new set of files. In this case, the DBSN of the old set of files is inserted into the DBSN of the new set, and the <sequence number> is inserted into the new files.

The default of <empty> provides the same results as RERUN but may be used only once against an old set of files; thereafter, it is prohibited. If it is desired to regenerate against such a set of files again, RERUN must be specified.

The <mode statement> indicates whether PRODUCTION or TEST files are being created. If PRODUCTION files are being created, this fact is displayed on the console printer and entered into the system log. The default value is PRODUCTION. If this is not the first run, this statement must be consistent with the old set of files being used. Test files are treated the same as production files except that no logging is done.

The <sequence number> is an integer between 0 and 99999999 and must be 1 greater than the number in the old set of files. If the highest number is reached, the sequence starts over again.

Example:

For a first production run:

```
SEQUENCECONTROL = FIRST, PRODUCTION, 1.
```

For a second run with the previous control files:

```
SEQUENCECONTROL = 2.
```

## UPDATEFMT

UPDATEFMT allows recompilation of the format section of the TCL deck without the need for a new generation. This may be done while the MCS is running. The update process involves the following steps:

1. The TCL source file that is to be updated should be the secondary input file "tape" to a "merge" run of the TCL compiler. Patch cards, including a patch card to the Control section to specify UPDATEFMT, should comprise the primary input file, thus identifying those areas of the format section that are to be updated. If file-equate cards were used for the original TCL generation, they should also be included in the update run.

When defines are used in formats or functions and the define is being patched, at least one card in the format or function that invokes the define should also be patched in order to recognize that the change is being made through a define. The card image in the format or function may be identical to the original card image.

2. Changes to old formats may be unit-tested by entering a Network Control command to put the appropriate station in a practice mode. While in practice mode, the patched format is invoked for that station, and word [3].[47:1] of the Common area has a value of 1 for all transactions entered on that station. All stations concurrently in data mode operate as though no patches were made.
3. For patched formats that have been tested, the patched format may be enforced on all stations by entering an UPDATE FORMAT Network Control command. A more complete description of the UPDATE FORMAT Network Control command is presented under UPDATE REQUEST in section 8.
4. The number of new formats compiled must fall within the MAXNEWFORMATS value specified in the most recent TCL generation. Each new format compiled in successive update runs decreases this value by one. New formats may be made visible to the MCS by entering an UPDATE DEVICE Network Control command. A more complete description of the UPDATE DEVICE Network Control command is presented under UPDATE REQUEST in section 8.

Although new functions may not be declared, updates to existing functions are unrestricted. New formats may be declared, and changes to existing formats are unrestricted. When function or format declarations span multiple card images, only those card images that need be changed are necessary in the update patch deck. New card images may be added in the appropriate card sequence.

## DEFINE SECTION

Syntax:

```
<define section> ::=
    DEFINE <definition list> . /
    <define section> <define section> /
    <empty>

<definition list> ::=
    <definition> / <definition> , <definition list>
```

```

<definition> ::=
    <define name>
    <optional define parameter list> = <text> #

<define name> ::= <identifier>

<optional define parameter list> ::=
    (<define parameter list>) / <empty>

<define parameter list> ::=
    <define parameter> /
    <define parameter>, <define parameter list>

<define parameter> ::= <identifier>

<text> ::=
    [Any sequence of valid characters except a # not in
    quotation marks.]

```

#### Semantics:

The DEFINE section causes the language processor to save off the specified <text> until such time as the <define name> is encountered in a <define invocation>. At that point, the saved <text> is inserted in place of the <define invocation>. Defines may be nested up to five (5) deep. They may only be declared between sections. The length of the <define name> is limited to 18 characters. Once a define is specified, it is valid throughout the rest of the TCL text.

A definition has two forms of syntax: the "simple" define and the "parametric" define. They are readily differentiated because the parametric define has a series of parameters (or <define parameter>s) enclosed in parentheses. A <define parameter> of one <definition> may not be used in the <define parameter list> of a parametric define invocation nested within it.

## DEFINE Invocation

#### Syntax:

```

<define invocation> ::= <define name> <actual text part>

<actual text part> ::= (<closed text list>) / <empty>

<closed text list> ::=
    <closed text> / <closed text>, <closed text list> /
    <empty>

<closed text> ::=
    [An actual text not containing unmatched parentheses
    or quotes, or not containing unbracketed commas.]

```

### Semantics:

The <define invocation> can be used within any section of the TCL. Its purpose is to minimize duplication of information where such duplication may be required. A define may be invoked only after it is declared.

A <define invocation> causes the <define name> to be replaced by the <text> associated with the <define name>. The invocation of a parametric define causes text substitution of the <closed text> into the indicated position(s) of the associated <text>. A <closed text> need not be "simple."

### Example:

```
DEFINE GROUT1 = AB1, AB2, XY37, ACD126#.
STATION MY/STATION (16):
VALIDUSERS = GROUT1
```

In the preceding example, valid-users for MY/STATION would be the four user-codes defined in GROUT1. GROUT1 could be invoked in other station descriptions as well.

### Example:

```
DEFINE FORMDEF (A,B,C) = A, V1:V1 OR B(C)#.
```

In the preceding parametric define example, an invocation for FORMDEF could be:

```
FORMDEF (A1, 6, (A15, "COST = $",A4))
```

which, if "expanded," would be:

```
A1, V1:V1 OR 6((A15, "COST = $",A4))
```

## GLOBAL SECTION

### Syntax:

<global section> ::= <global definition> / <empty>

<global definition> ::= GLOBAL: <global statement list>

<global statement list> ::=  
    <global statement> /  
    <global statement list>  
    <global statement>

```

<global statement> ::=
    <administrative program statement> /
    <batch io key statement> /
    <checkpoint interval statement> /
    <continuous processing statement> /
    <control audit statement> /
    <control permanent statement> /
    <controlled paging statement> /
    <format and function statement> /
    <format generator statement> /
    <log off message key statement> /
    <log on message key statement> /
    <master compute statement> /
    <max batch jobs statement> /
    <max new access keys statement> /
    <max new formats statement> /
    <max new mkes statement> /
    <max running statement> /
    <multiple station ident statement> /
    <my name statement> /
    <paging permanent statement> /
    <port max statement> /
    <queue block size statement> /
    <queue record size statement> /
    <recall program statement> /
    <service message ident statement> /
    <subsystems statement> /
    <system monitor statement> /
    <system monitor ident statement> /
    <system network control statement> /
    <system network control ident statement> /
    <system object io key statement> /
    <system prefix statement> /
    <zip compile statement> /
    <zip compile save statement>

```

Semantics:

The <global section> specifies information which pertains to general operation of the MCS. The data specified in this section applies to all systems defined in subsequent sections.

Example:

```

GLOBAL:
    CONTROLPERMANENT = TRUE.
    LOGONMKE = LOGON.
    LOGOFFMKE = LOGOFF.

```

## ADMINPROGRAM Statement

Syntax:

```
<administrative program statement> ::=
    ADMINPROGRAM =
        <program name>(<program number>)
        <global program classification> :
        <global program description>/
        <empty>

<global program classification> ::= USER / PROCESS / PORT

<global program description> ::=
    <global program statement>/
    <global program description>
    <global program statement>

<global program statement> ::=
    <program title statement> /
    <multiple inputs statement> /
    <permanent statement> /
    <common size statement> /
    <timeout statement> /
    <remarks statement> /
    <max copies statement> /
    <global input queue section>

<global input queue section> ::=
    INPUTQUEUE
    <input queue name>
    (<input queue number>) :
    <global input queue description>

<global input queue description> ::=
    <global input queue statement> /
    <global input queue description>
    <global input queue statement>

<global input queue statement> ::=
    <queue depth statement> /
    <time limit statement> / <empty>
```

Semantics:

The <administrative program statement> defines an administrative message switching program used to route messages from one station to another station, to a list of stations, or to an "area." The message key to be used by this program must be defined at the system level using the <administrative message key statement> syntax. A standard administrative message switching program is supplied with the GEMCOS System. Details on this program are presented in section 10.

## BATCHIOKEY Statement

Syntax:

```
<batch io key statement> ::=  
    BATCHIOKEY = <special character>. / <empty>
```

Semantics:

The <batch io key statement> is used to specify the message key appended to the handshake between a batch program and the MCS. If a BATCHIOKEY is not specified, batch programs may not be run.

Example:

```
BATCHIOKEY = *
```

## CHECKPOINTINTERVAL Statement

Syntax:

```
<checkpoint interval statement> ::=  
    CHECKPOINTINTERVAL = <integer>. / <empty>
```

Semantics:

The <checkpoint interval statement> specifies the amount of time, in seconds, which should elapse between checkpoints of the audit files.

The checkpoint consists of writing current information relative to the status of the Input and Output Queue files.

A word of caution is appropriate here: If checkpoints are taken too frequently, throughput can be impaired; if not frequent enough, recovery time can be increased unnecessarily.

If the statement is omitted, the interval is 30 seconds.

## CONTINUOUSPROCESSING Statement

Syntax:

```
<continuous processing statement> ::=  
    CONTINUOUSPROCESSING = <logical value>. / <empty>
```

Semantics:

The <continuous processing statement> specifies that the MCS is to run in a continuous-processing environment. If the statement is omitted, FALSE is assumed. For more information, refer to section 9.



## CONTROL AUDIT Statement

### Syntax:

```
<control audit statement> ::=  
    CONTROLAUDIT = <logical value>. / <empty>
```

### Semantics:

The <control audit statement> specifies whether input to the GEMCOS CONTROL stack will be audited in the input audit file. The default value is true (all inputs audited).

## CONTROLPERMANENT Statement

### Syntax:

```
<control permanent statement> ::=  
    CONTROLPERMANENT = <logical value>. / <empty>
```

### Semantics:

The <control permanent statement> specifies whether the Control program, a system-defined Process program, is to be run as a permanent or as a nonpermanent program. Refer to <permanent statement> in the Program section. If not specified, Control is nonpermanent.

## CONTROLLEDPAGING Statement

### Syntax:

```
<controlled paging statement> ::=  
    CONTROLLEDPAGING = <integer>. / <empty>
```

### Semantics:

The <controlled paging statement> specifies the maximum number of stations to be in Paging mode at any point in time.

This statement is intended to give the user some measure of control over the resources required to provide message paging.

Use of this statement does not preclude any station from working with paged messages, but it can delay a station once the maximum is reached.

If this statement is omitted, no limitation is placed upon the number of stations concurrently in Paging mode.

## FORMATGENERATOR Statement

Syntax:

```
<format generator statement> ::=  
    FORMATGENERATOR = <logical value>. / <empty>
```

Semantics:

The <format generator statement> is used to define the on-line FORMATGENERATOR program that allows a user to create and test formats and forms on-line from any station, transparent to the rest of the network. This program is available as a separately priced program product.

If this statement is TRUE, an entry is made in the program table as if the following TCL syntax had been declared:

```
PROGRAM FORMATGENERATOR (255) USER:  
    INPUTQUEUE FORMATMAKERQUEUE(255):  
    TITLE = GEMCOS/REFORM.  
    TIMEOUT = 600.  
    COMMONSIZE = 400.  
    AUDITINPUT = FALSE.  
    REMARKS = "HANDLES FORMAT MAKING CAPABILITIES."  
    FORMATMAKER = TRUE.
```

A terminal operator uses the FORMATGENERATOR program by assigning his or her station to the program and entering into a conversation with it. Refer to the GEMCOS Format Generator User/Reference Manual for a complete description of this program.

## QUEUERECORDSIZE Statement

Syntax:

```
<queue record size statement> ::=  
    <input/output queue record size statement> /  
    <input/output queue record size statement>  
    <queue record size statement>  
  
<input/output queue record size statement> ::=  
    INPUTQUEUERECORDSIZE = <integer>. /  
    OUTPUTQUEUERECORDSIZE = <integer>. / <empty>
```

Semantics:

The <queue record size statement> specifies the record size, in words, of the input and/or output queue disk files. The minimum size that may be specified is 30 words, and the maximum size is 300 WORDS.

The default record size for the Input Queue file is 30 words. The default record size for the Output Queue file is 60 words.

## LOGOFFMKE Statement

Syntax:

```
<log-off message key statement> ::=  
    LOGOFFMKE = <identifier>. / <empty>
```

Semantics:

The <log-off message key statement> specifies the message key used to remove an operator's access to a station following log-on to the station. This message key may be used only at stations where SIGNON = TRUE is specified. Additional information on access control is presented in section 7.

Example:

```
LOGOFFMKE = LOGOFF.
```

## LOGONMKE Statement

Syntax:

```
<log-on message key statement> ::=  
    LOGONMKE = <identifier>. / <empty>
```

Semantics:

The <log-on message key statement> specifies the message key which allows user access to a specified set of stations (refer to <valid user statement> in the STATION section) and a specified set of message keys (refer to <access control statement> in the SYSTEM section) when the internal MCS access control is being used. This message key may be used only at a station where SIGNON = TRUE is specified.

Example:

```
LOGONMKE = LOGON.
```

## MASTERCOMPUTE Statement

Syntax:

```
<master compute statement> ::=  
    MASTERCOMPUTE = <logical value>. / <empty>
```

Semantics:

The <master compute statement> specifies whether the compute function is usable by all stations in the network. Individual station usability can be controlled in the Station section. The compute function is discussed in detail in section 8. If this statement is not specified, FALSE is assumed.

## MAXBATCHJOBS Statement

Syntax:

```
<max batch jobs statement> ::=  
    MAXBATCHJOBS = <integer>. / <empty>
```

Semantics:

The <max batch jobs statement> specifies the maximum number of Batch jobs that may concurrently interface the MCS. The maximum that may be specified is 15 Batch jobs. If the statement is omitted, no Batch jobs may interface the MCS.

## MAXNEWACCESSKEYS Statement

Syntax:

```
<max new access keys statement> ::=  
    MAXNEWACCESSKEYS = <integer>. / <empty>
```

Semantics:

The <max new access keys statement> specifies the maximum number of new access keys that may be added through an UPDATE ACCESSKEY Network Control command before a new TCL generation is required. Default value is 5.

## MAXNEWFORMATS Statement

Syntax:

```
<max new formats statement> ::=  
    MAXNEWFORMATS = <integer>. / <empty>
```

Semantics:

The <max new formats statement> specifies the maximum number of new formats that may be added through an UPDATE DEVICE Network Control command before a new TCL generation is required. Default value is 5.

## MAXNEWMKES Statement

Syntax:

```
<max new mkes statement> ::=  
    MAXNEWMKES = <integer>. / <empty>
```

#### Semantics:

The `<max new mkes statement>` specifies the maximum number of new message keys or message IDs that may be added through an UPDATE MKE Network Control command before a new TCL generation is required. Default value is 5.

### MAXRUNNING Statement

#### Syntax:

```
<maxrunning statement> ::= MAXRUNNING = <integer> ./ <empty>
```

#### Semantics:

The `<max running statement>` specifies the number of Common areas through which User-type programs may run. This statement can be used to control the maximum number of User programs or copies of programs allowed to run concurrently. This statement must be specified and may have a value between 1 and 254.

### MULTIPLESTAIDENT Statement

#### Syntax:

```
<multiple station ident statement> ::=  
    MULTIPLESTAIDENT = <ident identifier> . / <empty>
```

```
<ident identifier> ::=  
    <identifier> / <integer> / <string>
```

#### Semantics:

The `<multiple station ident statement>` specifies the `<ident identifier>` to be placed at the end of the title line when a message is directed to more than one destination. The `<ident identifier>` may not be greater than 17 characters long. If this statement is omitted, MULTIPLE is assumed.

#### Example:

```
MULTIPLESTAIDENT = HOST.
```

### MYNAME Statement

#### Syntax:

```
<my name statement> ::= MYNAME = <identifier> . / <empty>
```

#### Semantics:

The `<my name statement>` causes the MYNAME attribute of the PORT file used by GEMCOS to communicate with PORTSTATIONS to be set to `<identifier>`. Every process which communicates with a PORTSTATION must set the YOURNAME attribute of its port subfile to `<identifier>`. Refer to the PORTSTATION statement in the station section for a more complete explanation. The default value of MYNAME is null.

## PAGINGPERMANENT Statement

Syntax:

```
<paging permanent statement> ::=  
    PAGINGPERMANENT = <logical value>. / <empty>
```

Semantics:

The <paging permanent statement> specifies whether the message paging module is to remain permanently in the mix. If it is FALSE or omitted, the module is executed when paging is to be performed and goes to EOJ when it has nothing left to do.

## QUEUEBLOCKSIZE Statement

Syntax:

```
<queue block size statement> ::=  
    QUEUEBLOCKSIZE = <integer>. / <empty>
```

Semantics:

The <queue block size statement> specifies the number of disk records preallocated per input queue and station when running in a continuous-processing environment.

Default value is 50, and maximum value is FILEAREASIZE (defined to be 4096). When CONTINUOUSPROCESSING is FALSE, QUEUEBLOCKSIZE has no effect. For more information, refer to section 9.

## RECALLPROGRAM Statement

Syntax:

```
<recall program statement> ::=  
    RECALLPROGRAM =  
        <program name>  
        (<program number>)  
        <global program classification>:  
        <global program description>
```

Semantics:

The <recall program statement> defines the Recall program used to recall any output message inserted into a queue either by its message number or by the date and time. The message key to be used by this program must be defined at the system level using the <recall message key statement> syntax. A standard Recall program is supplied with the GEMCOS system. Details on this and other auxiliary programs are presented in section 10.

## SERVICEMESSAGE IDENT Statement

Syntax:

```
<service message ident statement> ::=  
    SERVICEMESSAGE <ident identifier>. / <empty>
```

Semantics:

The <service message ident statement> specifies the <ident identifier> to be placed at the end of the title line when a service message is generated. The <ident identifier> may not be greater than 17 characters long. If this statement is not specified, SERVICEMESSAGE is assumed.

Example:

```
SERVICEMESSAGE = SERVICE/MESSAGE.
```

## SUBSYSTEMS Statement

Syntax:

```
<subsystems statement> ::=  
    SUBSYSTEMS: <specific list> / <empty>
```

```
<specific list> ::=  
    <specific item> /  
    <specific item> <specific list>
```

```
<specific item> ::=  
    CONTROL = <identifier> /  
    PAGER = <identifier> /  
    OUTPUT = <identifier> /  
    PROCESSEVERYTHING = <identifier> /  
    BACKUP = <identifier>
```

Semantics:

The purpose of this syntax is to allow the user to specify whether the GEMCOS stacks are to run out of a particular subsystem. For this syntax to be valid, the MCS and UTILITY must have been compiled with the SUBSYSTEMS option set. The system software used must allow the Subsystems task attribute to be used.

Example:

```
SUBSYSTEMS:  
    PAGER = LOCAL1.  
    CONTROL = LOCAL2.
```

## SYSTEMMONITOR Statement

Syntax:

```
<system monitor statement> ::=  
    SYSTEMMONITOR = <station name list>.
```

```
<station name list> ::=  
    <station name list> /  
    <station name list>, <station name>
```

```
<station name> ::= <generalized identifier>
```

Semantics:

The <system monitor statement> specifies the station or stations to receive messages that are of a system nature. Up to four stations can compose the <station name list>. This is a required statement, and at least one station must be declared as a system monitor. The <station name>s declared must be declared in the STATION section.

Example:

```
SYSTEMMONITOR = ANY/STA.
```

## SYSTEMMONIDENT Statement

Syntax:

```
<system monitor ident statement> ::=  
    SYSTEMMONIDENT = <ident identifier>. / <empty>
```

Semantics:

The <system monitor ident statement> specifies the <ident identifier> to be placed at the end of the title line when a message is directed to the system monitors. The <ident identifier> may not be greater than 17 characters long. If this statement is not specified, SYSMON is assumed.

Example:

```
SYSTEMMONIDENT = MONITOR.
```

## SYSTEMNETWORKCONTROL Statement

Syntax:

```
<system network control statement> ::=  
    SYSTEMSPO = <station name list>.
```



#### Semantics:

The <system network control statement> specifies the station or stations that can issue system control messages. Up to four stations can compose the <station name list>. This is a required statement, and at least one station must be declared as a System Network Control station. Unless AUTOENABLE is set to TRUE for a system, these stations are the only elements which are automatically enabled when the MCS initializes the network.

#### Example:

```
SYSTEMSPO = SPO/STA
```

### SYSTEMNETWORKCONTROLIDENT Statement

#### Syntax:

```
<system network control ident statement> ::=  
    SYSTEMSPOIDENT = <ident identifier>. / <empty>
```

#### Semantics:

The <system network control ident statement> specifies the <ident identifier> to be placed at the end of the title line when a message is directed to the System Network Control stations. The <ident identifier> may not be greater than 17 characters long. If this statement is not specified, SYSSPO is assumed.

#### Example:

```
SYSTEMSPOIDENT = SPOS.
```

### SYSTEMOBJECTIOKEY Statement

#### Syntax:

```
<system object io key statement> ::=  
    SYSTEMOBJECTIOKEY = <special character>. / <empty>
```

#### Semantics:

The <system object io key statement> specifies the key preceding each message to and from an external Object program communicating with the MCS. If the statement is not specified, external object programs may not communicate with the MCS.

#### Example:

```
SYSTEMOBJECTIOKEY = &.
```

## SYSTEMPREFIX Statement

Syntax:

```
<system prefix statement> ::=  
    SYSTEMPREFIX = <ident identifier> / <empty>
```

Semantics:

The <system prefix statement> specifies the <ident identifier> used to prefix the system name in service messages. The <ident identifier> may not be longer than 17 characters.

Example:

```
SYSTEMPREFIX = MICHIGAN.
```

## FORMAT AND FUNCTION SECTION

Syntax:

```
<format and function statement> ::=  
    <format section> / <empty>  
  
<format section> ::=  
    <function declaration list>  
    <format declaration list>  
  
<function declaration list> ::=  
    <function declaration> /  
    <function declaration list> <function declaration> /  
    <empty>  
  
<format declaration list> ::=  
    <format declaration> /  
    <format declaration list> <format declaration>
```

Semantics:

The formats and related constructs for input and output formatting are described in this section.

## FUNCTION Declarations

Syntax:

```
<function declaration> ::= FUNCTION <function part list>.  
  
<function part list> ::=  
    <function part> /  
    <function part list>, <function part>
```

```

<function part> ::=
    <function designator>
    <justification and fill part> (<translation list>)

<function designator> ::= <identifier>

<justification and fill part> ::=
    [EXTERNAL: <function type>
    INTERNAL:<function type>] /
    <empty>

<function type> ::= INTEGER / ALPHA / UNEDITED

<translation list> ::=
    <translate pair> /
    <translation list>, <translate pair>

<translate pair> ::= <external string>:<internal string>

<external string> ::=
    [Any string of no more than six characters]

<internal string> ::=
    [Any string of no more than six characters]

```

#### Semantics:

Functions are used with formats. They allow a format to translate a string of length N into a string of length M where  $0 < N < 7$  and  $0 < M < 7$ .

<External string>s and <internal string>s are limited to a maximum of six characters each.

On input, an <external string> is translated into the associated <internal string>. On output, an <internal string> is translated into its associated <external string>.

The <external string>s and <internal string>s are stored in a pair of associated arrays. The mask-search intrinsic is used to search the appropriate array and return a string from the associated array.


Examples of functions used in formats are shown following the section on the semantics of <format declaration>s.

Examples:

```
FUNCTION GENDER (" MALE": "1", "FEMALE": "2").
FUNCTION DAY ("1": "SUN", "2": "MON", "3": "TUES", "4": "WED", "5": "THU",
"6": "FRI", "7": "SAT").
FUNCTION NUM1 ("ONE": "01", "TWO": "02", "THREE": "03", "FOUR": "04",
"FIVE": "05", "SIX": "06", "SEVEN": "07", "EIGHT": "08", "NINE": "09", "TEN":
"10", "ELEVEN": "11")
FUNCTION NUM2 [EXTERNAL: ALPHA, INTERNAL: INTEGER] ("ONE": "01",
"TWO": "02", "THREE": "03", "FOUR": "04", "FIVE": "05", "SIX": "06", "SEVEN":
"07", "EIGHT": "08", "NINE": "09", "TEN": "10", "ELEVEN": "11").
```

If the <justification and fill part> is <empty>, it is assumed that both the <external string>s and <internal string>s are unedited. By using the <justification and fill part>, the user may make either of these strings unedited, integer, or alpha.

An unedited string of less than six characters in length is right-justified within a 6-character word with leading nulls (4"00").

An integer string of less than six characters is right-justified with  trailing zeros.

An alpha string of less than six characters is left-justified with trailing blanks.

Within a given function, as long as all <internal string>s are of the same length and all <external string>s are of the same length, it makes little difference whether the strings are unedited, integer, or alpha. However, if strings vary in length, the use of integer or alpha strings can help to avoid confusion. Suppose a function is declared as follows:

Example:

```
FUNCTION TEST ("1": "ME", "11": "SOME").
```

On input, a search for an <external string> of "1" would result in a match with "11" because its right-most position contains "1" and the mask-search looks only at bits [7:8]. A similar phenomenon would occur on output with a search for an <internal string> of "ME", which would result in a match with "SOME". (The mask-search always starts at the end of an array and searches backward). This problem could be avoided by declaring the function as follows:

```
FUNCTION TEST
[EXTERNAL: INTEGER, INTERNAL: ALPHA]
("1": "ME", "11": "SOME").
```

Now, if asked to search for "1", the mask-search would actually search for "000001", and if asked to search for "ME", would look for "ME ".

## FORMAT Declarations

Syntax:

<format declaration> ::= FORMAT <format part list>.

<format part list> ::=  
    <format part> /  
    <format part list> <format part>

<format part> ::=  
    <format designator> <format specification>

<format specification> ::=  
    LIBRARY / LIBPAGED /  
    <special action part> (<local declaration part>  
    <line count statement> <format description>)

<format designator> ::= <identifier>

<special action part> ::= [<special action list>] / <empty>

<special action list> ::=  
    <special action> /  
    <special action list>, <special action>

<special action> ::=  
    UPDATE <input specification>/RESIDENT/ SEQUENTIAL

<input specification> ::=  
    ,INPUT : <format designator> / <empty>

<local declaration part> ::=  
    VARIABLE <variable declaration list>; / <empty>

<variable declaration list> ::=  
    <variable declaration> /  
    <variable declaration>, <variable declaration list>

<variable declaration> ::=  
    <variable identifier>  
    <optional location specifier> FOR <integer>

<variable identifier> ::= V1/V2/V3/V4/V5/V6

<optional location specifier> ::=  
    <location specifier> / <empty>

<line count statement> ::= LINECOUNT = <integer>; / <empty>

<format description> ::=  
     <editing specifications> / <page format list>

<page format list> ::=  
     <page format> / <page format list>, <page format>

<page format> ::=  
     PAGE [<page number>] <page repeat>:  
     <editing specifications>

<page number> ::=  
     <integer> / <page end phrase> \* <page begin phrase>

<page end phrase> ::= <simple string list>

<page begin phrase> ::= <simple string list>

<simple string list> ::=  
     <simple string> / <simple string>, <simple string list>

<simple string> ::=  
     <EBCDIC code> <EBCDIC string> /  
     <hexadecimal code> <hexadecimal string>

<page repeat> ::=  
     REPEAT ON <REM or variable list> / <empty>

<REM or variable list> ::=  
     <REM or variable> /  
     <REM or variable>, <REM or variable list>

<REM or variable> ::= REM / <variable identifier>

<editing specifications> ::=  
     <editing segment> /  
     <editing specifications>, <editing segment>

<editing segment> ::= <fixed segment> / <variable segment>

<variable segment> ::= [<fixed segment>]

<fixed segment> ::= <editing phrase list>

<editing phrase list> ::=  
     <editing phrase> /  
     <editing phrase list>, <editing phrase>

```

<editing phrase> ::=
    <editing string> / <location specifier> /
    <item phrase> / <case statement> /
    <set variable statement> /
    <header declaration> / <break variable statement> /
    <end-of-line statement>

<editing string> ::= <skip field> / <simple string>

<skip field> ::= X<integer> / X(<variable field delimiter>)

<variable field delimiter> ::= <one-byte string>

<location specifier> ::=
    @ <integer> / @ + <integer> / @ - <integer>

<item phrase> ::=
    <repeat part> (<fixed segment>) /
    <repeat part> <item phrase part> /
    T(<function designator>, <item phrase part>,
    <internal size>)

<repeat part> ::=
    <integer> /
    <update variable> <REM or variable> OR <integer> /
    <empty>

<update variable> ::= <variable identifier>: / <empty>

<item phrase part> ::=
    <item type> <field width> / <F item type>

<item type> ::= A / I / B / J / C / L

<field width> ::=
    <integer> / (<variable field specifier>)

<F item type> ::=
    <optional sign> <optional dollar> <optional comma>
    F <integer> . <integer> /
    <optional dollar> <optional comma> F <integer> .
    <integer> <optional sign>

<optional sign> ::= S / <empty>

<optional dollar> ::= $ / <empty>

<optional comma> ::= P / <empty>

```

`<internal size> ::= <integer>`  
  
`<variable field specifier> ::=`  
`<variable field delimiter>, <internal size>`  
  
`<case statement> ::=`  
`CASE <optional variable identifier>`  
`(<statement lists> <optional else>)/`  
`<empty>`  
  
`<optional variable identifier> ::=`  
`<variable identifier> / <empty>`  
  
`<statement lists> ::=`  
`<statement list> / <statement lists>, <statement list>`  
  
`<statement list> ::= <integer> : (<editing phrase list>)`  
  
`<optional else> ::= ELSE: (<editing phrase list>) / <empty>`  
  
`<set variable statement> ::=`  
`<variable identifier> FOR <integer> / <empty>`  
  
`<header declaration> ::=`  
`H<integer> (<heading string list>) /`  
`H<integer> / <empty>`  
  
`<heading string list> ::=`  
`<heading string> /`  
`<heading string> <heading string list>`  
  
`<heading string> ::=`  
`<simple string list> /`  
`<repeat part> <item type> <field width>`  
  
`<break variable statement> ::=`  
`BREAK<variable identifier> / <empty>`  
  
`<end-of-line statement> ::= </> / <empty>`

Semantics:

`<format declaration>`

The `<format declaration>` is used for declaring a format to be applied to a message on its way into or out of the MCS. In general, a format is applied to an input message or an output message but not both. An exception to this is the update format which is discussed under `<special action>`. Which format is applied to which message is determined by statements in the DEVICE section.



### <format part list>

The <format part list> allows several formats to be described in one <format declaration>. Each <format part> describes a particular message format.

### <special action list>

A <special action part> after the <format identifier> may indicate that the format is a resident format, an update format, a sequential format, or any combination thereof.

A resident format is kept in an array instead of on disk. This facility is provided for small, frequently used formats. It is intended to save the I/O overhead that would otherwise be required to retrieve a format from disk before using it. This feature should be used with care since its overuse could require significant amounts of main memory.

An update format is one that displays a formatted message on a screen device and allows the terminal operator to change unprotected fields and transmit the screen back to the MCS. This feature has meaning only for paged formats. When a screen is formatted, fields can be re-arranged. When the screen is returned to the MCS, it must be put back in the same sequence that it was in before formatting. You might say it must be "unformatted." For any update format, the system automatically "unformats" the message when returned. There are two important restrictions which must be observed when declaring an update format: 1) <editing string>s (refer to Syntax for FORMAT DECLARATIONS) may be used only in untransmittable fields (outside the scope of forms characters), and 2) <editing string>s must not be used within transmittable fields. (See the discussion below on the use of the C and L <item type>s to alleviate these restrictions.)

The <input specification> feature allows the user to specify an input format to be applied to the message after all of the update paging dialogue is completed (after the X paging dialogue character has been transmitted and the "unformat" has been applied). This feature is only allowed for update formats. It enables the user to reduce the amount of data being written to the input audit and sent to the program. The <format identifier> must be previously specified in the TCL syntax.

A sequential format may be used only on messages whose fields are in the proper sequence. Absolute or negative location specifiers (@) cannot be used in a sequential format. For long messages which are in the proper sequence, the MCS does not have to read the entire message from disk before formatting it. This feature is intended to allow the user to avoid the memory overhead that might otherwise be associated with a large, paged message.

### <local declaration>

A <local declaration part> may be included in a nonupdate output <format declaration> to declare variables that are used in a <repeat part> within a <format declaration>. Values for variables declared are the first items extracted from the internal message. During each variable assignment, the "internal message pointer" is adjusted by a combination of the <optional location specifier> and the internal character length of the variable as specified in the variable declaration. Variable values contained in the internal message should be expressed as EBCDIC characters with a value not greater than 255.

### <line count statement>

The <line count statement> defines the line count for a paged format. The definition must occur at the beginning of the format and can have any value from 1 to 255. The line count may be defined only for a paged update or display format. Once it is defined, a line count work area is established; when the defined line maximum is met, a page break occurs, clearing the line count work area to zero.

### <page begin-end phrase>

A <page begin-end phrase> may be defined at the beginning of the format, but is only allowed on update or display paged formats. The definition may be composed of strings only. A format with this definition always begins "broken" terminal outputs with the PAGE BEGIN string. It also appends the PAGE END string to all "broken" terminal outputs. The <page begin-end phrase> is invoked by the <break variable statement> syntax or when the line count work area reaches the specified line maximum.

Example:

```
FORMAT ABC (LINECOUNT = 24;
PAGE[4"27E60000",*, 4"0C0000", "[+]", / ]:
    4"0C0000", "[+]", 4"0D25", / ,
    A6,4"0D25", 1,
    V1 FOR 2,
    V1 OR 20 (I6, 4"0D25",/),
    V2 FOR 2,
    BREAKV2,
    V2 OR 20 (A8, 4"0D25",/),
    4"27E60000", "X").
```

### <format description>

A <format description> consists of <editing specifications> or, in the case of a paged format, page numbers interspersed with <editing specifications>. (Refer to <editing specifications> below.)

### <REM or variable list>

Variables and update variables are used to represent repeated occurrences of data. They are initially assigned values from the user TP message and can be counted down by using update variables. When used in a <page repeat>, that page of a format is repeated as long as the variables are greater than zero.

"REM", used in the <page repeat> and the <repeat part>, refers to an internal reserved variable that has meaning after any repeat expression. Its use is primarily to provide for a "total line" capability. The "total line" is to be executed after all repeats are exhausted. A specific definition of REM is provided below:

REM = IF (L-V)<0 THEN 0 ELSE (L-V)

where L is the literal maximum of a repeat, and V is a variable. For example, a repeat expression might be:

V1 or 12

If V1 were greater than 12, the repeat expression above would be executed the literal maximum of 12. All 12 of the repeat applications would be used, and REM would be 0 after the repeat. If V1 were 9, however, the repeat would be terminated by V1, and in this case, would be 3 after the repeat.

### <editing specifications>

The <editing specifications> consist of one or more <editing segment>s separated by commas. Each <editing segment> can be fixed or variable.

### <variable segment>

A <variable segment> is a <fixed segment> surrounded by brackets. In order to be meaningful, at least two <editing phrase>s within a <variable segment> must have a <variable field specifier> for the <field width>. This feature allows the user to describe a message which contains two or more fields, the sequence of which is unknown; however, they may be identified by the <1-byte string> delimiting them. The exact meaning of this is made clearer by the example of the system format file in appendix B, under SYSTEM FORMAT FILES.

### <fixed segment>

A <fixed segment> is an <editing phrase list> which can be surrounded by parentheses and preceded by a <repeat part>. The <repeat part> may be an unsigned integer (which simply means that the <editing phrase list> is repeated <integer> times) or it may be a variable repeat expression. If the value of the variable is zero when a repeat is encountered, editing phrases bounded by the repeat are skipped. A variable repeat expression must include a maximum repeat count which is used to terminate the repeat if the variable count has not been reached. When an <update variable> is specified, the <update variable> is assigned the value of the variable, decremented by one for each repeat, until the repeat is terminated.

### <editing phrase list>

An <editing phrase list> is one or more <editing phrase>s separated by commas. An <editing phrase> can be one of three things: 1) an <editing string>, 2) a <location specifier>, or 3) an <item phrase>.

### <editing string>

An <editing string> is either a <simple string>, such as "XYZ" or 4"F12B", or a <skip field> such as X3 or X(","). When a <simple string> is encountered during the application of a format, it is inserted into the formatted message. When a <skip field> is encountered on input, spaces are skipped in the incoming, raw data message. X3 causes three spaces to be skipped. X(" ,") causes spaces to be skipped until a comma is encountered in the raw data message. When a <skip field> is encountered on output, it causes blanks to be inserted in the outgoing, formatted message. X3 would cause three blanks to be inserted. X(<variable field delimiter>) is undefined for output messages.

### <location specifier>

A <location specifier> is used to manipulate the "internal message pointer." On input, it manipulates a pointer to the formatted message; on output, it manipulates a pointer to the raw, unformatted message. The <location specifier> should be envisioned as referencing the Common array as a message is in the process of being moved in (input) or out (output). This internal pointer normally moves one character at a time to the right as <editing phrase>s are processed. The <location specifier> allows it to be "manually" set or adjusted. When a sign is present, it is adjusted forward or backward by <integer> characters. If no sign is present, it is set to the position specified by the <integer>. Care must be taken on input not to overlay good data when using this feature.

### <item phrase>

<Item phrase>s come in a variety of forms, all of which involve an <item type>. There are seven <item type>s:

- A : ALPHA
- B : TABBED ALPHA
- C : ALPHA – OUT ONLY
- F : FIXED FIELD
- I : INTEGER
- J : TABBED INTEGER
- L : ALPHA – IN ONLY LOCATION SPECIFIER-OUT ONLY

There are basically two differences between alpha fields and integer fields: 1) Integer fields are edited to make sure that they contain only digits or blanks (no imbedded blanks), and 2) Justification and blank/zero fill are different. On input, if a source alpha field is smaller than the destination, the data is left-justified with trailing blanks. On input, if an integer is smaller than its destination integer field (regardless of the size of the source field), the integer is right-justified with leading zeroes. This means that a 3-digit integer may appear anywhere within a 10-digit field on input and is right-justified with leading zeroes in the formatted message. On output, the use of a <variable field specifier> as the <field width> of an <item phrase> results in data compression by removal of trailing blanks in alpha fields and leading zeroes or blanks in integer fields. In both cases, the <delimiter> is inserted after the compressed field in the formatted message.

The simplest form of an <item phrase> is one with a fixed field size such as A6 or I9. An A6 <item phrase> results in the unedited move of six characters from the raw message to the formatted message. An I9 <item phrase> moves nine characters and is subject to the editing rules mentioned in the preceding paragraph.

B and J <item phrase>s are the same as A and I phrases, respectively, except that on input, a field in the incoming message can end early if a tab character (4"05") is encountered. Thus, in a B10 phrase in an input message, characters are transferred until either 10 characters have been moved or a tab character is encountered. If the transfer is terminated by a tab character, trailing blanks are added to the field in the formatted message in order to fill out all 10 characters. On output, B and J behave exactly the same as A and I.

When a <variable field specifier> is used in an input message in A or I fields, a variable length field delimited by <variable field delimiter> (which can be any <one-byte string>) is moved into a field of <internal size> characters in the formatted message. A <variable field specifier> used with B or J fields on input allows the user to substitute any <one-byte string> for the normal tab character (4"05"). Thus, B(4"05",10) is the same as B10.

The C <item type> is for use in update formats only. It is intended for use in alphanumeric data fields in untransmittable areas on a screen device (not within forms characters). Since other <item types> may be used only in transmittable fields, the C <item type> is necessary to allow critical data to be shown to the terminal operator and returned to the TP unchanged in an update transaction on a screen device.

The F <item type> allows editing of numeric fields on input and output. The allowable characters in this field depend on the syntax selected for the field; alpha characters and blanks are never allowed. Justification and zero fill follow the same rules as for I (integer) fields.

The complete syntax for the F field takes one of two forms:

- a. <S> <\$> <P> FW.D
- b. <\$> <P> FW.D <S>

Delimiters (< >) indicate the syntactic item is optional. In the first form, S indicates a leading sign, while in the second form it indicates a trailing sign. The \$ indicates a leading dollar sign. P indicates that comma insertion is to occur on output and comma scan/deletion is to occur on input. F is the <item type>. W indicates an integer that represents the total field width; however, W does not include the number of commas and the decimal point. The period acts as a delimiter between the integers W and D. D indicates an integer less than or equal to W and represents the number of decimal places in the field.

The L <item type> is for use only in update formats to relieve the restriction that <editing string> may not be declared within transmittable fields (within the scope of forms characters). Strings may be inserted into transmittable fields only if the transmittable field which contains the string is followed by an L <item type> whose field length is exactly the same as the length of the string.

#### NOTE

The paging dialogue character in a transmittable field should not be reflected by a corresponding L field.

Any of the <item phrase>s discussed thus far may be repeated by using a <repeat part> in front of the <item type>.

The T(<function designator>, <item type> <field width>, <internal size>) form of <item phrase>, when used on an input message, translates a field in the incoming message described by <item type> <field width> into a field of the same type and <internal size> characters long in the formatted message. The translation is defined by the <function designator> which was previously declared as a function. This provides a capability similar to that provided by a translate table in ALGOL. The function can be declared to translate any N characters into any M characters where 0<N<7 and 0<M<7. On output, a field the width of <internal size> is translated into a field described by <item type> <field width>.

### <case statement>

The <case statement> allows the selection of editing phrases based on a variable or on the two characters beginning at the current internal pointer location in the message. When the <selection expression> is <empty>, the CASE assumes the value of these next two characters. If the ELSE form of the <statement list> is used, it must be the last of the <statement list>s. If no ELSE <statement list> is used, a format error occurs when an undefined value is used as the <selection expression>.

The <case statement> may be used in output, input, paged update, or display formats. The <selection expression> (either type) of the <case statement> is not sent to the terminal on output and is re-established from the original message on input (unformat) of paged update.

Example:

```
FORMAT OUTONLY (VARIABLE V1 FOR 2;
CASE V1 (1: (A6, "CASE 1")
        2: (A3, "CASE 2")
        ELSE: ("CASE 3"))).

OUTPUT MESSAGE "02ABC"

SENT TO TERMINAL "ABCCASE 2"

FORMAT INONLY (A4,
CASE (01: (A6, "CASE 1")
      02: (A3, "CASE 2")
      ELSE: ("CASE 3"))). -

INPUT FROM TERMINAL "MKE101ABCDEF"

EDITOR'S MESSAGE TO TP "MKE1ABCDEF CASE 1"
```

### <set variable statement>

A variable can be set in any spot in the format. The <integer> represents the number of character positions from the current pointer location that are integerized to establish the variable. The <integer> can be from 1 to 3 digits, and the maximum value of the variable is 255. The characters used to set the variable are not sent to the terminal. When update paging is used during the input (unformat) phase, the variable is re-established from the internal message.

Example:

```
FORMAT VAR (A3,
V1 FOR 3, V1 OR 10 (I1, 4"0D")).

OUTPUT MESSAGE "ABC00297"

SENT TO TERMINAL "ABC9<CR>7<CR>"
```

### <header declaration>

A format sequence may be defined as a header. Any number of headers may be nested, up to a level of 15. The strings are saved, and they precede the message after the page break. Header levels are defined by the integer following the H in the syntax. A header of a given level replaces any previously declared header of the same level and deletes all headers of greater levels. HN-deletes all headers greater than or equal to that level.

Example:

```
FORMAT XXX [UPDATE] (LIVECOUNT=10;
PAGE(4"27E60000",*,4"000000","[+] RESULTS CONTINUED",
4"0025",/);
4"000000","[+] PATIENT TEST RESULTS",4"0025",/,
H1("PATIENT NAME",4"00",/),
V1 FOR 2,
V1 OR 20(H2(C15,3+5,4"00",/),
H3(" TEST NUMBER TEST RESULTS",4"00",/),
V2 FOR 2,
V2 OR 20(CASE(01:("[" ,A4,"] [" ,A6,"]
[" ,A2,"]" , 4"00",/)
02:("[" ,A4,"] [" ,A1,"] [" ,A10,"]
[" ,A2,"]" ,4"00",/))
H2-),
4"27E60000","X").
```

### <break variable statement>

The <break variable statement> causes a test of certain conditions and invokes a page break when required. A comparison of VM to the line-count work area can be done by the syntax BREAKVM. If VM plus the line-count work area is greater than the line-count definition, a page break occurs.

### <end-of-line statement>

An end-of-line character has been defined as `"/"`. This character causes the line-count work area to be incremented by 1.

Example:

```
FORMAT ABC (LINECOUNT = 24;  
PAGE[4"27E60000",*,4"0C0000","[+]",/]: %"BEGIN-END"  
4"0C0000","[+]",4"0D25",/  
A6,4"0D25"/"  
V1 FOR 2,  
V1 OR 20(I6,4"0D25,/),  
V2 FOR 2,  
BREAKV2,  
V2 OR 20(A8,4"0D25",/),  
4"27E600","X").
```

In this example, if V1 were 15 and V2 were 23, the following would occur:

1. The screen would be cleared, a `"[+]"` would appear on the first line followed by a carriage return/linefeed (CRLF) and the line-count work area would be counted up to 1.
2. One line of A6 format would be applied, the CRLF inserted, and the line-count work area would be counted up to 2.
3. The value of V1 should be extracted from the message, and V1 would be 15.
4. Fifteen lines of I6 followed by CRLF would be executed, and the line-count work area would be incremented to 17.
5. The value of V2 would be extracted from the message, and V2 would be 23.
6. V2 would be added to the line-count work area and the sum (40) would be found to be greater than the line count (24).
7. A page break would occur with all strings before the `"**"` applied to the current text before transmitting, and all strings after the `"**"` applied to the beginning of the next text to be transmitted. The line-count work area would be set to zero and then counted up to 1.
8. Twenty lines of A8 followed by CRLF would be executed and the line-count work area would be counted up to 21.



## Editing Errors

When errors are detected on output, the message is sent to its normal destination, and an error message is sent to a system monitor specifying what type of error occurred.

When an error is detected on input, the "standard editor" sets the format error bit in COMMON. If necessary, the user may obtain more information about the error by changing the editor or writing another. Additional information may be obtained in the following manner:

```
0:      % NON-DIGIT IN INTEGER FIELD
1:      % TARGET OF TRANSLATE FUNCTION NOT FOUND
2:      % DELIMITER MISSING OR VARIABLE FIELD TOO LONG
3:      % ATTEMPT TO USE VARIABLE SEQUENCE ON OUTPUT
4:      % FORMATTED MESSAGE NEEDS LARGER DESTINATION ARRAY
5:      % INVALID VARIABLE ASSIGNMENT VALUE
6:      % LOCATION SPECIFIER OUT OF BOUNDS
7:      % MESSAGE TOO LONG
8:      % INVALID FIXED FORMAT
10:     % REPEAT INDEX TOO LARGE FOR PAGE BREAK
64:     % TOO MANY DELIMITERS FOUND IN VARIABLE SEQUENCE
65:     % INVALID OP CODE - COULD BE USING WRONG FORMAT
        % FILE
66:     % UNDEFINED DELIMITER
67:     % INVALID CASE NUMBER
101:    % FAULT IN FORMATTER: ZERODIVIDE
102:    % FAULT IN FORMATTER: EXPONENTOVERFLOW
103:    % FAULT IN FORMATTER: EXPONENTUNDERFLOW
104:    % FAULT IN FORMATTER: INVALIDINDEX
105:    % FAULT IN FORMATTER: INTEGEROVERFLOW
107:    % FAULT IN FORMATTER: MEMORYPROTECT
108:    % FAULT IN FORMATTER: INVALIDOP
109:    % FAULT IN FORMATTER: LOOP
110:    % FAULT IN FORMATTER: MEMORYPARITY
111:    % FAULT IN FORMATTER: SCANPARITY
112:    % FAULT IN FORMATTER: INVALIDADDRESS
113:    % FAULT IN FORMATTER: STACKOVERFLOW
114:    % FAULT IN FORMATTER: STRINGPROTECT
115:    % FAULT IN FORMATTER: PROGRAMMEDOPERATOR
116:    % FAULT IN FORMATTER: INVALIDPROGRAMWORD
ELSE:   % ROOM FOR FUTURE EXPANSION
END;
```

Thus RESULT.[0:1]=1 IF AN ERROR WAS DETECTED  
RESULT.[7:7]= ERROR NUMBER  
RESULT.[23:16]=CHARACTER POSITION IN MESSAGE  
              OF OFFENDING CHARACTER OR LOCATION SPECIFIER  
RESULT.[39:16]=LOCATION OF DESTINATION  
              POINTER WHEN ERROR OCCURRED  
RESULT.[47:8]=OP CODE UNDER EXECUTION WHEN  
              ERROR OCCURRED

Figures 3-1 and 3-2 provide graded examples of increasing complexity of formats applied to input messages and output messages. Figure 3-3 provides more complex examples involving variable field sizes, sequenced fields, and tabs. Figure 3-4 provides examples of formats involving translate functions.

Input/ Output -----	Message As It Appears at the Terminal -----	<Editing Specifications> Applied to Message In Transit -----	Message As It Appears to the User Program -----
Input/ Output	SHAKESPEARE	A11	SHAKESPEARE
Input/ Output	1066	I4	1066
Input/ Output	SPECIAL#*/	A10	SPECIAL#*/
Input/ Output	ADM63902	A3,I5	ADM63902
Input	ADM 63902	A3,I7	ADM0063902
Input	ADM63902	A3,I7	ADM0063902
Input/ Output	ALPHAGAMMA	A1,	ALPHAGAMMA
Input/ Output	ALPHAGAMMA	A5,A5	ALPHAGAMMA
Input/ Output	ALPHAGAMMA	2A5	ALPHAGAMMA
Input/ Output	AB123XY456	A2,I3,A2,I3	AB123XY456
Input/ Output	AB123XY456	2(A2,I3)	AB123XY456
Input	AB123XY456	I10	AB123XY456 (error flag set)
Input	AB123XY456	I12	0CAB123xy456 (error flag set)

Figure 3-1. Graded Examples of <editing specifications> Applied to Input and Output Messages

Input/ Output -----	Message As It Appears at the Terminal -----	<Editing specifications> Applied to Message In Transit -----	Message As It Appears to the User Program -----
Input	DOE,JOHN	A3,X1,A4	DOE JOHN
Input/ Output	DOE JOHN	A3,X1,A4	DOE JOHN
Output	DOE,JOHN	A3,"",A4	DOEJOHN
Input	DOE,JOHN	A3,"",A4	DOE,,JOH
Input	RIGHT	A5,8"FACE"	RIGHTFACE
Output	RIGHTFACE	A5,8"FACE"	RIGHT
Output	NAME: [HARRY]C 2	"NAME: [",A5,"]",4"12"	HARRY
Input/ Output	ALPHA	a3,A5	ALPHA
Input/ Output	DOE,JOHN	a5,A4,a1,A4	JOHNDOE,
Input	DOE,JCHN	a5,A3,X1,a1,A4	JOHNDOE
Input/ Output	AB123XY456	A2,a5,I3,a3,A2,a8,I3	ABXY123456
Output	02/10/75	I2,2(""/",I2)	021075
Input	A123485678	2(A1,2(X1,I1))	A24868
Input/ Output	A 2 48 6 8	2(A1,2(X1,I1))	A24868
Input	1234XY	I6	1234XY (error flag set)
Output	1234XY (monitor notified of error)	I6	1234XY
Input/ Output	+ \$123,456.78	S&PF8.2	12345678
Input/ Output	2.1-	F3.1S	-021

Figure 3-2. Graded Examples of <editing specifications> Applied to Input and Output Messages

Input/ Output -----	Message As It Appears at the Terminal -----	<Editing specifications> Applied to Message In Transit -----	Message As It Appears to the User Program -----
Input/ Output	15P	I("P",5)	00015
Input/ Output	ES	A("S",3)	E
Input/ Output	ES15P	A("S",3),I("P",5)	E 00015
Input/ Output	DOE, JOHN	A(",",6),X1,A6	DOE JOHN
Input	SHAKESPEARE, BILL	A(",",6),X1,A6	SHAKESBILL (error flag set)
Input	12345*	I("*",4)	2345 (error flag set)
Input	T A1B2A B	B6	A1B2
Input	WASHINGTONGEORGE	B10,B12,A6	WASHINGTON GEORGE
Input	T DOEAJOHN B	B10,B12,A6	DOE JOHN
Input	SHAKESPEAREBILL	B10,B12,A6	SHAKESPEAR EBILL
Input	T SHAKESPEARABILL B	B10,B12,A6	SHAKESPEAR EABILL B
Input	T T SHAKESPEAREABILLA B B	B10,B12,B6	SHAKESPEAR E
Input	DOE,JOHN	B(",",10),B12,A6	DOE JOHN
Input	WASHINGTONGEORGE	B(",",10),B12,A6	WASHINGTON GEORGE

Figure 3-3. More Complex Examples: Variable Field Sizes, Sequenced Fields, and Tabs (Sheet 1 of 2)

Input/ Output -----	Message As It Appears at the Terminal -----	<Editing specifications> Applied to Message In Transit -----	Message As It Appears to the User Program -----
Input	WASHINGTON,GEORGE	B(" ",10),B12,A6	WASHINGTON ,GEORG
Input	SHAKESPEARE,BILL	B(" ",10)B12,A6	SHAKESPEAR E,BILL
Output	DOE [5815042]	C6,"[",A7,"]"	DOE 5815042
Input	T 1234A5678 *	J6,I6,A1	001234005678*
Input	B 12345678 *	J("5",6),I6,A1	001234000678*
Input	DOE8JOHNA	[A("A",6),A("B",6)]	JOHN DOE
Input	DOEAJOHNB	[A("B",6),A("A",6)]	JOHN DOE
Input	DOEAJOHNB	[A("B",6),X3,A("A",6)] NOTE:X3 PHRASE ABOVE IGNORED	JOHN DOE
Input	ABC#123456*X	[A("#",6),I3,I(" ",4)],A1	ABC 1230456X
Input	456*ABC#123X	[A("#",6),I3,I(" ",4)],A1	ABC 1230456X
Input	456*RSTUVABC#12 X	[A("#",6),I3,I(" ",4)],A1	TUVABC0120456X
Input	456*RSTUVABC#12 X	[A("#",6),I3,B11,I(" ",4)],A1	TUVABC012 0456X

Figure 3-3. More Complex Examples: Variable Field Sizes, Sequenced Fields and Tabs  
(Sheet 2 of 2)

Input/ Output -----	Terminal -----	<Editing Specifications> -----	User Program -----
Input	ONE MALE	T(NUM1,A3,1), T(GENDER,A6,1)	11
Input/ Output	FOURFEMALE	T(NUM1,A4,1), T(GENDER,A6,1)	42
Input/ Output	TWO3	T(NUM2,A3,1),T(DAY,A1,3)	2TUE
Input	SIX#X	T(NUM2,A("#",6),2),A1	06X
Input	ELEVEN#X	T(NUM2,A("#",6),2),A1	11X
Input	TWENTY#X	T(NUM2,A("#",6),2),A1	??X (error flag set)
Output	(FOUR)	"(",T(NUM2,A("#",6),2)	04 )
Input	WED	T(DAY,A3,1)	? (error flag set)
Output	??? (monitor notified)	T(DAY,A3,1)	2 )
Input/ Output	ONE X	T(NUM2,A6,1),A1	1X
Output	ELEVENX	T(NUM1,A6,1),A1	1X
Input/ Output	MALE	T(GENDER,A6,1)	1
Input	MALE	T(GENDER,A4,1)	2

Figure 3-4. Examples of Formats Involving Translate Functions

The functions used may be found in the examples given in the description of functions in this section.

The last two pairs of examples in figure 3-4 illustrate a problem that can arise when function elements vary in length and are identical in their right-most positions. This problem may be avoided by using the <justification and fill part> as in NUM2; this results in the mask-search examining all six characters, including leading zeros and trailing blanks.

## Formatting of MCS Error Responses

Certain MCS error responses may be reformatted by device class by means of output formatting. For each error response provided below, the MCS inserts an output-message-id into the control words array of the output message. If the error output-message-id is associated with a format for the device in question, the error response is formatted accordingly before being released to the station. The error output-message-IDs are:

ID	Meaning
SECERR	Security error
INVMKE	Invalid message key
BUSY	Input received while station is in transaction mode.
TRNERR	Transmission error
UPDDIS	Updates disabled message
LONOK	Log-on accepted messages
LOFOK	Log-off accepted messages

### Pragmatics:

It is helpful when working with formats to think of the MCS as shown in figure 3-5.

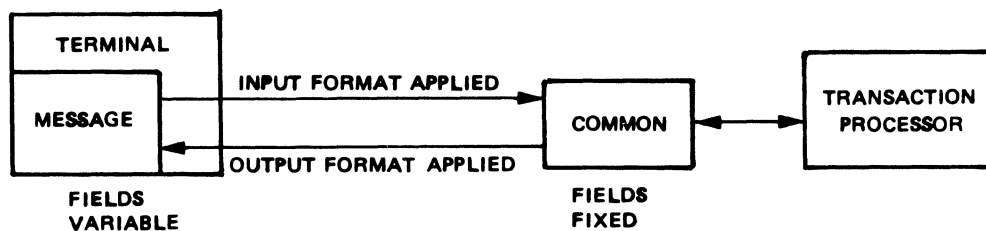


Figure 3-5. Interrelationship of Fixed Fields in Common and Possible Variable Fields at Terminal

The format is meant to describe the relationship between the fixed fields in common and the corresponding and possibly variable fields at the terminal.

When a format is applied to a message, data fields can be moved, expanded, or compressed. Fields can be moved in such a way as to change their order. For fields described as integer fields (I editing phrase), leading zeros are inserted (when necessary) on input. For alphanumeric fields (A editing phrases), trailing blanks are inserted on input when necessary.

## Field Sequence

To understand field sequencing, think of the message as existing in two forms: one at the terminal and one in Common. Imagine a pointer into each form of the message as shown in figure 3-6.

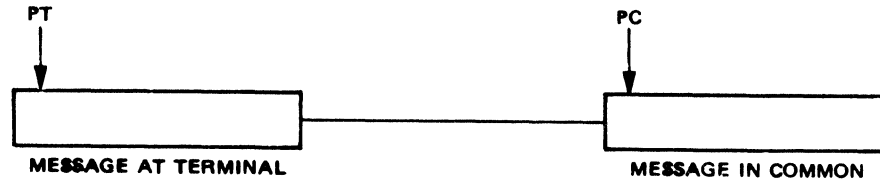


Figure 3-6. Terminal and Common Message Pointers

The fields in the message at the terminal can be variable in length and even variable in sequence (on input only). The fields in the message in Common must be fixed in both length and sequence.

The pointers PT and PC in figure 3-6 both begin pointing at the first character position (position 1) in their respective messages. As editing phrases in a format are applied to data fields, the data is moved from one message to the other, and these pointers are updated. Unless specifically instructed to do otherwise, these pointers are updated by moving them to the right by the number of characters moved. For instance, suppose an input message of "ABC 123" is applied to a format which begins with an A4 editing phrase. After applying the A4, the situation that would then exist is shown in figure 3-7. PT may be advanced without affecting PC by using X editing phrases. (PT, in general, may be moved only to the right). Given the situation shown in figure 3-7, the comma in the message can be skipped over with an X1 editing phrase.

PC may be moved in either direction without affecting PT by using a location specifier (@<unsigned integer>). In figure 3-7, PC is now pointing at position 5. An @8 editing phrase may be used to advance to position 8 before transferring any more characters. Figure 3-8 is an updated picture of figure 3-7 after applying an X1 and an @8 editing phrase.

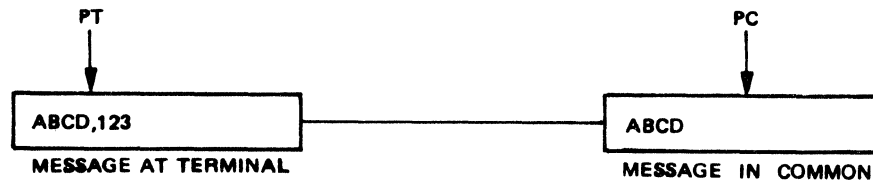


Figure 3-7. Illustration of Pointer Updates



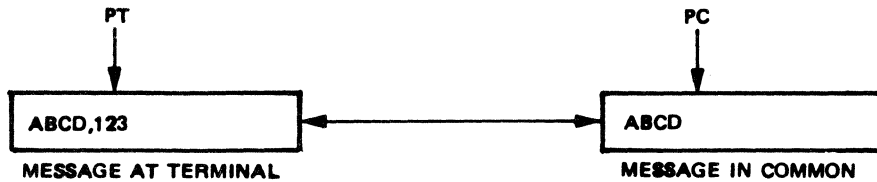


Figure 3-8. Pointer Update After Applying an X1 and an @8 Editing Phrase

Finally, an I3 editing phrase is processed. Figure 3-9 shows the final status of the messages and their respective pointers after applying the editing specifications: A4, X1, @8, and I3.

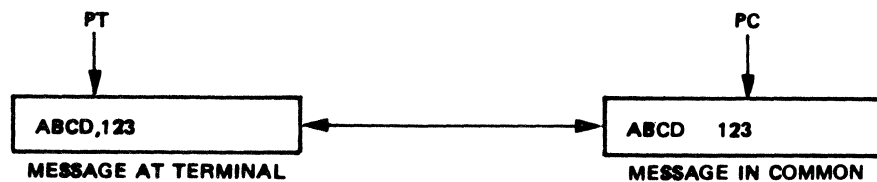


Figure 3-9. Pointer Update After Applying Specifications A4, X1, @8, and I3

## SYSTEM SECTION

Syntax:

```

<system section> ::=
    SYSTEM <system name> (<system number>):
    <system description>

<system name> ::= <identifier> / <string> / <integer>

<system number> ::= <integer>

<system description> ::= <system statement list>

<system statement list> ::=
    <system statement> /
    <system statement list> /
    <system statement>

<system statement> ::=
    <access control statement> /
    <administrative message key statement> /
    <archival audit statement> /
    <autoenable statement> /
    <auto recovery statement> /
    <clear restart statement> /
    <flush recovery statement> /
  
```

```

    <host statement> /
    <recall message key statement> /
    <subsystem statement> /
    <empty>

```

Semantics:

The SYSTEM section specifies information relative to an individual system. This section is required if CONTROL = GENERATE is specified in the GLOBAL section.

The <system name> specifies the name of the system. It is used in output titles and service messages, if they were not defined in the GLOBAL section.

The <system number> may not be a duplicate of another <system number>.

A maximum of eight systems may be declared.

Example:

```

SYSTEM SOS (1):
    ADMINKEY = ADM.
    RECALLKEY = RECALL.

```

## ACCESSCONTROL Statement

Syntax:

```

<access control statement> ::=
    ACCESSCONTROL = <association list>. / <empty>

```

```

<association list> ::=
    <association> /
    <association list> <association>

```

```

<association> ::= ACCESSKEY <access code> =
    <access mke list>.

```

```

<access code> ::= <identifier>

```

```

<access mke list> ::= <mke list> / ALL

```

```

<mke list> ::= <mke> / <mke list> <mke>

```

#### Semantics:

The <access control statement> specifies to the system the valid users of the system and the message keys each valid user is allowed to enter into the system. If ALL is specified, this user may input any message key without restriction. For more information on ACCESSCONTROL, refer to section 7.

#### Example:

```
ACCESSCONTROL =  
  ACCESSKEY AB1234 = INQ, DX6, XYZ.  
  ACCESSKEY AB5678 = INQ, X14, P67.
```

### ADMINKEY Statement

#### Syntax:

```
<administrative message key statement> ::=  
  ADMINKEY = <identifier>. / <empty>
```

#### Semantics:

The <administrative message key statement> defines the message key which is used to identify administrative messages.

If this syntax is used, an administrative program must be defined in the GLOBAL section. Specifications for the GEMCOS-supplied administrative program are presented in section 10.

All messages headed by this administrative message key are routed to the administrative program.

### ARCHIVALAUDIT Statement

#### Syntax:

```
<archival audit statement> ::=  
  ARCHIVALAUDIT = <logical value>. / <empty>
```

#### Semantics:

ARCHIVALAUDIT = TRUE allows a selective period dump to tape of all messages in the audit files related to all ARCHIVALAUDIT = TRUE or RECOVERY = TRUE queues for archival storage. If the statement is omitted, FALSE is assumed.

Whenever the TCL files are generated or regenerated with the <status> parameter of the <sequence control statement> not equal to FIRST (refer to GLOBAL section of TCL syntax), or CONTINUOUS PROCESSING is TRUE and a ?DUMP DATACOM Network Control command is entered, the old Input and Output queues are dumped to a tape in a form which can be used by Archival Recall and by Archival Recovery.

#### NOTE

The network, the on-line TPs, and GEMCOS CANNOT be active during the regeneration phase.

For each archival dump, an entry is made in a ledger file. Each entry in the file contains the date, sequence-control number of that generation of files, and the serial numbers of the tapes used to store the archival dump. If, during the regeneration of the TCL files, the utility does not find a ledger file, it creates a new one; however, to maintain the integrity of the archival recovery mechanism and the archival retransmission of messages, the same ledger file should be present for all TCL regenerations. This ensures that an accurate history of the network is maintained. The internal name of the ledger file is ADTLEGFILE; the file may be file-equated. The layout of the ledger file is contained in Appendix B.

## AUTOENABLE Statement

Syntax:

```
<autoenable statement> ::=
    AUTOENABLE = <logical value. / <empty>
```

Semantics:

AUTOENABLE = TRUE specifies that all input queues, programs and stations in this system will be automatically enabled when the MCS is run. If FALSE is specified, then the operator must enter an ENABLE system control message to allow the MCS to start processing transactions for this system. The default value is FALSE.

## AUTORECOVERY Statement

Syntax:

```
<auto recovery statement> ::=
    AUTORECOVERY = <logical value>. / <empty>
```

Semantics:

AUTORECOVERY = TRUE indicates that recovery is initiated automatically, following a TP failure. If the failure occurs in transaction state, the TP is first automatically disabled and cleared; then synchronized recovery is initiated. When recovery is complete, all programs, input queues, and stations in the system are automatically enabled. When AUTORECOVERY = FALSE, a DISABLE, CLEAR, and subsequent ENABLE must be operator-initiated.

If a TP fails outside of transaction state, or if a TP fails while processing a nonsynchronized recoverable transaction, the TP is allowed to run again. If the user does not want the program to run again, the TP's TASKVALUE.[39:8] should be set to high values. The default value is FALSE.

## CLEARRESTART Statement

Syntax:

```
<clear restart statement> ::=
    CLEARRESTART = <logical value>. / <empty>
```

Semantics:

The <clear restart statement> specifies whether a system's RESTART PROGRAM is to be initiated when starting up GEMCOS to clear the Restart data sets. This is not done if the prior termination of the MCS resulted from a Halt/Load. If this statement is not specified, FALSE is assumed.

Pragmatics:

CLEARRESTART = TRUE insures that there are no extraneous records in the restart data set which reflect some prior condition of the input and output audit files. If this option is not set to TRUE, it is the user's responsibility to insure that the restart data set is empty after each queue generation or regeneration.

## FLUSHRECOVERY Statement

Syntax:

```
<flush recovery statement> ::=
    FLUSHRECOVERY = <logical value>. / <empty>
```

Semantics:

FLUSHRECOVERY = TRUE indicates that no recovery of any kind is performed for this system after a TP abort or a system halt/ load (refer to section 5).

## HOST Statement

Syntax:

```
<host statement> ::= HOST = <identifier>. / <empty>
```

Semantics:

The <host statement> specifies the <host name> attribute to be used when initiating the Port transaction processing programs in the system. In order to use the <host statement>, the PORTS compile time option must be SET when compiling the MCS and UTILITY. The <identifier> may be changed by a Network Control command.

## RECALLKEY Statement

Syntax:

```
<recall message key statement> ::=  
    RECALLKEY = <identifier>. / <empty>
```

Semantics:

The <recall message key statement> defines the message key used to route a recall message to the Recall program.

If this syntax is used, a Recall program must be defined in the GLOBAL section. Specifications for the GEMCOS-supplied Recall program are presented in section 10.

Example:

```
    RECALLKEY = RECALL.
```

## SUBSYSTEM Statement

Syntax:

```
<subsystem statement> ::= SUBSYSTEM = <identifier>. / <empty>
```

Semantics:

The <subsystem statement> specifies the <identifier> attribute to be used in initiating all Port transaction processing programs in this system. This <identifier> may be changed by a Network Control command. The MCS and the utility must both be compiled with the SUBSYSTEMS \$ compile option set.

## PROGRAM SECTION

Syntax:

```
<program section> ::= <program define list>
```

```
<program define list> ::=  
    <program define> /  
    <program define list> <program define>
```

```
<program define> ::=  
    PROGRAM <program name> (<program number>)  
    <program classification>:  
    <program description>
```

<program name> ::= <identifier>

<program number> ::= <integer>

<program classification> ::=  
 USER / PROCESS / PORT / SERVICE / EDITOR

<program description> ::= <program statement list>

<program statement list> ::=  
 <program statement> /  
 <program statement list> <program statement>

<program statement> ::=  
 <access control program statement> /  
 <AP300 status statement> /  
 <chargecode statement> /  
 <common size statement> /  
 <control bit statement> /  
 <conversationsize statement> /  
 <declared priority statement> /  
 <editor statement> /  
 <family statement> /  
 <formatmaker statement> /  
 <host statement> /  
 <inactive timeout statement> /  
 <invalid messages statement> /  
 <max copies statement> /  
 <may not be assigned statement> /  
 <min copies statement> /  
 <modify statement> /  
 <multiple inputs statement> /  
 <permanent statement> /  
 <remarks statement> /  
 <restart program statement> /  
 <return messages statement> /  
 <service statement> /  
 <subspaces statement> /  
 <subsystem statement> /  
 <test program statement> /  
 <timeout statement> /  
 <title statement> /  
 <transmission error message statement> /  
 <usercode statement> /  
 <input queue section>

Semantics:

The PROGRAM section is the means by which data communications tasks are defined to the MCS. The programs are either User programs which run in the mix or Process programs which are a part of the data communications system.

If there are any nonpermanent programs declared in the TCL, the PROGRAM section may not define more than  $\langle \text{max running} - 1 \rangle$  User programs with PERMANENT = TRUE, because each permanent program is assigned a row of Common which is the communication link between User programs and the MCS.

A  $\langle \text{program define} \rangle$  must contain a  $\langle \text{program name} \rangle$ ,  $\langle \text{program number} \rangle$   $\langle \text{program classification} \rangle$ , and  $\langle \text{title statement} \rangle$ . If the  $\langle \text{program classification} \rangle$  is User, Port, or Process, then the definition must also contain an  $\langle \text{input queue section} \rangle$  and a  $\langle \text{timeout statement} \rangle$ . User and Port programs must also contain a  $\langle \text{common size statement} \rangle$ .

The  $\langle \text{program name} \rangle$  and  $\langle \text{program number} \rangle$  may not be a duplicate of another  $\langle \text{program define} \rangle$ . A  $\langle \text{program number} \rangle$  may range from 1 to 98 or from 100 to 255. Program number 99 is the Control program. If the Format Generator is declared in the GLOBAL Section, it is assigned Program number 255.

The  $\langle \text{program classification} \rangle$  defines the interface which the program uses with the data communications system. The five classifications (User, Process, Port, Service, and Editor) are described in section 4.

A Service program must be defined before it is referenced by a  $\langle \text{service statement} \rangle$  in a User program. An Editor program must be defined before it is referenced by an  $\langle \text{editor statement} \rangle$  in a User or Port program.

A Process program may not use a Service or Editor program. A Port program may not use a Service program. Not all  $\langle \text{program statement} \rangle$ s are valid for a given  $\langle \text{program classification} \rangle$ . See individual  $\langle \text{program statements} \rangle$ s for restrictions.



Example:

```
PROGRAM FIR(1) SERVICE:
  TITLE = PRG/DC/FIR.
  CONTROLBIT = 2.
PROGRAM EDIT(2) EDITOR:
  TITLE = PRG/DC/EDITOR.
  CONTROLBIT = 3.
PROGRAM DRIVE (98) USER:
  TITLE = 1D99.
  COMMONSIZE = 300.
  MULTIPLEINPUTS = TRUE.
  PERMANENT = TRUE.
  SERVICE = FIR.
  EDITOR = EDIT.
  TIMEOUT = 60.
  INPUTQUEUE IQU1(1):
    MKE = MKE(1,1), MKE2(1,2).
  REMARKS = "DRIVER INQUIRY PROGRAM".
PROGRAM TP1(1) PORT:
  TITLE = MYPORT/PROG.
  INPUTQUEUE IQU2(2):
    MKE = MKE3(1,3), MKE4(1,4).
  TIMEOUT = 60.
  PERMANENT = TRUE.
```

## ACCESSCONTROLPROGRAM Statement

Syntax:

```
<access control program statement> ::=
  ACCESSCONTROLPROGRAM = <logical value>. / <empty>
```

Semantics:

The <access control program statement> specifies whether this program handles all sign-on and sign-off requests from stations in this system.

The MCS routes all <sign on> messages to this program. The program, in turn, must return either an indicator or an access key which the MCS uses to validate subsequent message keys that are input from the station.

If the statement is omitted, FALSE is assumed.

#### Pragmatics:

Only one program within a system may have ACCESSCONTROLPROGRAM = TRUE. The message keys described for this program are considered to be sign-on or sign-off message keys by the MCS.

If a system has no program description containing ACCESSCONTROLPROGRAM = TRUE, default sign-on control prevails. Default is that sign-on consists of <log-on message key> and user code for all stations with SIGNON = TRUE. The interface between the MCS and an access control program is described in section 7.

An ACCESSCONTROLPROGRAM is the only program that can send ? Network control commands to the Control program via the TP-TO-TP mechanism.

#### NOTE

The Access Control module need not return an access control error to the MCS but may send a message directly back to the station.

### AP300STATUS Statement

#### Syntax:

```
<AP300 status statement> ::=  
    AP300STATUS = <logical value>. / <empty>
```

#### Semantics:

When a program is specified as AP300STATUS = TRUE, it will receive all messages from any stations in the same system which have TYPE = AP300. Only one program per system may have AP300STATUS = TRUE. The default value is FALSE.

### CHARGECODE Statement

#### Syntax:

```
<chargecode statement> ::=  
    CHARGECODE = <identifier>. / <empty>
```

#### Semantics:

The <chargecode statement> specifies the CHARGECODE under which this program will run. If the statement is omitted, the program is run with no CHARGECODE.

#### Example:

```
CHARGECODE = MYCHARGE.
```

## COMMONSIZE Statement

### Syntax:

```
<common size statement> ::=  
    COMMONSIZE = <integer>. / <empty>
```

### Semantics:

The <common size statement> specifies the length in words of the Common area used to interface this program with the MCS. This statement is required for all User and Port programs. The size must be greater than 23 and less than or equal to 4000. The <common size statement> may not be specified for Process, Editor, or Service programs.

### Example:

```
COMMONSIZE = 300.
```

## CONTROLBIT Statement

### Syntax:

```
<control bit statement> ::=  
    CONTROLBIT = <integer>. / <empty>
```

### Semantics:

The <control bit statement> specifies the bit number used to indicate when an Editor or Service program has control of the Common area. Acceptable values range from 2 to 255. The control bit value for each Service and Editor program must be unique, and the <control bit statement> may not be specified for Process, User, or Port programs. The control-bit value of a User or Port program is always 0. The control-bit value for the MCS is 1.

### Example:

```
CONTROLBIT = 10.
```

## CONVERSATIONSIZE Statement

### Syntax:

```
<conversation size statement> ::=  
    CONVERSATIONSIZE = <integer>. / <empty>
```

#### Semantics:

The `<conversation size statement>` specifies the size of the program's Conversation area in words. The Conversation area is part of the Common area beginning at word 23 (the beginning of the text area for nonconversational programs) and extending for `<conversation size>` words. The Conversation area is null on the first message of the day and on the first message after an End-of-Conversation. Text immediately follows the Conversation area, and word 7 of Common, the length word, does not include the length of the Conversational data. See the `<conversational statement>` in the STATION SECTION and a more detailed description of the CONVERSATION feature in Section 4.

### DECLAREDPRIORITY Statement

#### Syntax:

```
<declared priority statement> ::=
    DECLAREDPRIORITY = <integer>. / <empty>
```

#### Semantics:

The `<declared priority statement>` specifies the system task priority that is used by the MCS when the task is initiated. The `<integer>` may be changed by a Network Control command. Default value is 60.

### EDITOR Statement

#### Syntax:

```
<editor statement> ::= EDITOR = <program name>. / <empty>
```

#### Semantics:

The `<editor statement>` specifies the Editor program that this program may use. An Editor program must be defined prior to its appearance in an `<editor statement>`. An `<editor statement>` may not appear in a Service, Editor, or Process program definition.

#### Example:

```
EDITOR = EDIT.
```

### FAMILY Statement

#### Syntax:

```
<family statement> ::=
    FAMILY [any valid MCP syntax for the
    family attribute]. / <empty>
```

#### Semantics:

The `<family statement>` allows a family attribute to be attached to the program when it is initiated by the MCS. The string must be valid MCP syntax for the family attribute. If no family is specified, the task is initiated with the same family as the MCS.

#### Example:

```
FAMILY DISK = MYPACK OTHERWISE DISK.
```

### FORMATMAKER Statement

#### Syntax:

```
<formatmaker statement> ::=  
    FORMATMAKER = <logical value>. / <empty>
```

#### Semantics:

FORMATMAKER = TRUE specifies that this program is a format generator. Refer to the `<format generator statement>` in the GLOBAL section and to the B B 6000/B 7000 Series GEMCOS Format Generator User/Reference manual, form 1121340, for more information. The default value is FALSE.

#### Example:

```
FORMATMAKER = TRUE.
```

### HOST Statement

#### Syntax:

```
<host statement> ::= HOST = <identifier> / <empty>
```

#### Semantics:

The `<host statement>` specifies the `<host name>` attribute to be used when initiating this transaction processing program. In order to use the `<host statement>`, the PORTS compile time option must be TRUE. This statement is valid only for a Port program definition.

#### NOTE

When the `<host statement>` is declared at the program level, synchronized recovery may be impacted.

The `<identifier>` may be changed by a Network Control command.

## INACTIVETIMEOUT Statement

Syntax:

```
<inactive timeout statement> ::=
    INACTIVETIMEOUT = <integer>. / <empty>
```

Semantics:

The <inactive timeout statement> defines the amount of time (in seconds) that may elapse without any activity from this program before considering it inactive. The value of this statement may be between 1 and 4095 and is only valid for User, Port, or Process programs. When a User, Port, or Process program has no input for this amount of time, the MCS automatically sends the program to EOJ. A value of 4095 causes the program to remain in the mix permanently.

If this statement is omitted, the default value is 4095.

## INVALIDMESSAGES Statement

Syntax:

```
<invalid messages statement> ::=
    INVALIDMESSAGES = <logical value>. / <empty>
```

Semantics:

The <invalid messages statement> specifies whether the program is to receive all messages which are unrecognizable to the MCS, i.e., messages which do not contain a <mke> defined in any program's <mke list> and which are not among the system-defined input messages. Only one program may be defined with INVALIDMESSAGES = TRUE. This program must be a Process program.

### NOTE

There need not be an INVALIDMESSAGES = TRUE program. When the program is not specified, the MCS responds to invalid messages by printing out an error message.

Example:

```
INVALIDMESSAGES = TRUE.
```

## MAXCOPIES Statement

Syntax:

```
<max copies statement> ::= MAXCOPIES = <integer>. / <empty>
```

#### Semantics:

The <max copies statement> specifies the maximum number of copies of this program that may be executed concurrently. The dynamic execution of additional copies is controlled by the <queue depth statement> and the <time limit statement>. A maximum of seven copies of a program may be declared in this statement. This maximum may be changed by means of a Network Control command, but it may never be made greater than the value specified here.

If this statement is omitted, one copy is assumed.

### MAYNOTBEASSIGNED Statement

#### Syntax:

```
<may not be assigned statement> ::=  
    MAYNOTBEASSIGNED = <logical value>. / <empty>
```

#### Semantics:

The <may not be assigned statement> specifies if a station should be allowed to assign itself to this program. This implies certain protocols and agreements between the station and program.

Value TRUE would indicate that the station may not be assigned. If not specified, FALSE is assumed.

### MINCOPIES Statement

#### Syntax:

```
<min copies statement> ::= MINCOPIES = <integer>. / <empty>
```

#### Semantics:

The <min copies statement> specifies the number of copies of this program that will remain in the mix despite the queue depth and time limit criteria for multiple copies. The <min copies statement> may only be used if the <max copies statement> is defined. Default is 0.

### MODIFY Statement

#### Syntax:

```
<modify statement> ::= MODIFY = <logical value>./ <empty>
```

Semantics:

The <modify statement> specifies that the program modifies a data base and that all transactions to the program may be denied when a Network Control command is entered to disable updates. A subsequent Network Control command to enable updates re-allows all transactions to the program. If the statement is not specified, FALSE is assumed. The <modify statement> may not be specified for a Service or Editor program.

Example:

```
MODIFY = TRUE.  
MODIFY = FALSE.
```

## MULTIPLEINPUTS Statement

Syntax:

```
<multiple inputs statement> ::=  
    MULTIPLEINPUTS = <logical value>. / <empty>
```

Semantics:

The <multiple inputs statement> specifies whether the program can accept more than one message per BOJ. The program goes to EOJ when no more messages are available to process (unless the program is specified to be permanent). The <multiple inputs statement> may not be specified for a Service or Editor program. If the statement is omitted, FALSE is assumed.

Example:

```
MULTIPLEINPUTS = TRUE.  
MULTIPLEINPUTS = FALSE.
```

## PERMANENT Statement

Syntax:

```
<permanent statement> ::=  
    PERMANENT = <logical value>. / <empty>
```

Semantics:

The <permanent statement> specifies if the program stays in the mix after BOJ. If PERMANENT = TRUE, the program remains in the mix until system EOJ, until a control message forces an EOJ, or until the INACTIVETIMEOUT expires. The program must be a MULTIPLEINPUTS = TRUE program. A NOINPUT = TRUE program may not be a permanent program. The <permanent statement> may not be specified for a Service or Editor program. If the statement is omitted, FALSE is assumed.

Example:

```
PERMANENT = TRUE.  
PERMANENT = FALSE.
```



## REMARKS Statement

Syntax:

<remarks statement> ::= REMARKS = <remarks string>./ <empty>

<remarks string> ::= <string> / <remarks string>, <string>

Semantics:

The <remarks statement> specifies a <string> that is to be saved for documentation purposes. CONTROL = LIST is used to print out this listing. The maximum length of a <remarks string> is 100 characters. The <remarks statement> is not required.

Example:

```
REMARKS = "DRIVER INQUIRY PROGRAM".
```

## RESTARTPROGRAM Statement

Syntax:

<restart program statement> ::=  
 RESTARTPROGRAM = <logical value>./  
 <empty>

Semantics:

The <restart program statement> identifies the program responsible for providing restart data to the MCS during recovery. One program with RESTARTPROGRAM = TRUE must be declared for each system which has a RECOVERY = TRUE input queue.

If the statement is omitted, FALSE is assumed. Detailed information regarding recovery is presented in section 5.

## RETURNMESSAGES Statement

Syntax:

<return messages statement> ::=  
 RETURNMESSAGES = <logical value>. / <empty>

Semantics:

The <return messages statement> specifies whether the program is to receive all messages returned to the MCS by a ROUTEHEADER = TRUE station. Only a Process program may be defined with RETURNMESSAGES = TRUE.

Example:

```
RETURNMESSAGES = TRUE.
```

## SERVICE Statement

Syntax:

<service statement> ::= SERVICE = <program name> ./ <empty>

Semantics:

The <service statement> specifies the Service program that a User program may use. A Service program must be defined prior to its appearance in a <service statement>. A <service statement> may not appear in a Service, Editor, Port, or Process program definition. The Service program interface is discussed in detail in section 4.

Example:

```
SERVICE = FIR.  
SERVICE = FMR.
```

## SUBSPACES Statement

Syntax:

<subspaces statement> ::= SUBSPACES = <integer> . / <empty>

Semantics:

The <subspaces statement> specifies the system subspaces attribute that is established by the MCS when the task is initiated. The <integer> must be a valid subspaces value, and may be changed by a Network Control command.

## SUBSYSTEM Statement

Syntax:

<subsystem statement> ::=  
SUBSYSTEM = <identifier> . / <empty>

Semantics:

The <subsystem statement> specifies the <identifier> to be used as the SUBSYSTEM attribute when initiating this transaction processing program. The <identifier> may be changed by a Network Control command. The MCS and the utility must both be compiled with the SUBSYSTEMS \$ compile option set.

## TESTPROGRAM Statement

Syntax:

```
<test program statement> ::=  
    TESTPROGRAM = <logical value>. / <empty>
```

Semantics:

The <test program statement> is used to specify whether this program is in test status. When TESTPROGRAM = TRUE, the MCS strips the <message key> from all input messages routed to this program.

This method of program testing offers flexibility because test versions of programs can be seen "side-by-side" with their production-status counterparts.

If the <test program statement> is omitted, FALSE is assumed.

Pragmatics:

The message keys for test programs are really dummy message keys designed for routing messages to the test programs. Messages must also contain the REAL message-key immediately following the test message-key.

The MCS performs routing based upon the test message-key and removes it from the message prior to passing the message to the program. The program receives the message just as if it were in the Production mode. The fact that the program is in TEST mode is passed to the program in the COMMON area or in the CONTROLWORDS array.

Example:

```
Station inputs: TST6 ABC123 <message text>  
MCS does routing based upon "TST6"  
MCS passes: "ABC123 <message text>" to test program
```

## TIMEOUT Statement

Syntax:

```
<timeout statement> ::= TIMEOUT = <integer>. / <empty>
```

Semantics:

The <time-out statement> defines the amount of time (in seconds) that must elapse before the MCS considers the program as having timed out after a pass of control to the program by the MCS. The <timeout statement> may have a value between 1 and 4095 inclusively and is required for all User, Port, and Process programs. If the program does not respond within this amount of time, a nonfatal warning is sent to the system monitors.

## TITLE Statement

Syntax:

```
<title statement> ::=  
    TITLE = <optional user code>  
           <generalized identifier>  
           <pack specification>  
  
<optional user code password> ::= (<user code>) / <empty>  
  
<pack specification> ::= ON <pack id> / <empty>  
  
<user code> ::= <identifier>
```

Semantics:

The <title statement> specifies the name of the program to be executed. This statement is required for all programs. The optional user code associated with the program is printed on the TCL listing but is not printed on any directory listing. The user code is printed on a table listing. The optional user code is used by GEMCOS when processing the TP; it is not used by GEMCOS in any way as part of the access control mechanism for transaction routing or sign-ons. The use of user code in the title affects only the finding of the code file of this program by GEMCOS and does not mean that the program will run under this usercode. The <usercode statement> causes the program to run under a particular usercode.

Example:

```
TITLE = 1D99.  
TITLE = PRG/DC/1VO1 ON MYPACK.  
TITLE = (USERCODE)TEST/PROG.
```

## TRANSMISSIONERRORMESSAGES Statement

Syntax:

```
<transmission error message statement> ::=  
    TRANSMISSIONERRORMESSAGES = <logical value>. / <empty>
```

#### Semantics:

The <transmission error message statement> specifies whether the program is to receive messages from the MCS that indicate an error has occurred during input from a station. These messages are usually responded to with a message indicating that the station should resend the message since it had a transmission error. Only one program may be defined with TRANSMISSIONERRORMESSAGES = TRUE. This program must be a Process-type program.

#### NOTE

A transmission error message program need not be defined for a system. In this case, the MCS automatically sends a warning message to the station when transmission errors occur.

#### Example:

```
TRANSMISSIONERRORMESSAGES = TRUE.
```

### USERCODE Statement

#### Syntax:

```
<usercode statement> ::= USERCODE = <identifier> ./ <empty>
```

#### Semantics:

The <usercode statement> specifies the usercode under which the program will run. If it is omitted, the program will run under the same usercode as GEMCOS.

#### Example:

```
USERCODE = MYUSER.
```

### INPUTQUEUE SECTION

#### Syntax:

```
<input queue section> ::= <input queue define list>
```

```
<input queue define list> ::=  
    <input queue define> /  
    <input queue define list> <input queue define>
```

```
<input queue define> ::=  
    INPUTQUEUE <input queue name> (<input queue number>):  
    <input queue description>
```

```

<input queue name> ::= <identifier>

<input queue number> ::= <integer>

<input queue description> ::=
    <input queue statement list> /
    <no input statement>

<input queue statement list> ::=
    <input queue statement> /
    <input queue statement list> <input queue statement>

<input queue statement> ::=
    <archival audit statement> /
    <audit statement> /
    <audit input statement> /
    <memory limit statement> /
    <message key statement> /
    <queue depth statement> /
    <recovery statement> /
    <time limit statement> /
    <wait for audit statement> /
    <empty>

```

#### Semantics:

The <input queue section> is used to define the Input queues through which User, Port, and Process programs run. Up to three Input queues per User or Port program may be defined, but only one Input queue may be defined for a Process program. They are processed from a low to a high order of priority in the order that they are defined. The highest priority queue for a program must be empty before GEMCOS looks for other input for that program.

If a <no input statement> is specified, only one queue may be specified, and this must be the only statement in the <input queue section>.

The <input queue name> and <input queue number> may not be a duplicate of another <input queue define>. An <input queue number> may range from 1 to 999. The <input queue section> may be defined only for User, Port, or Process programs.

## ARCHIVALAUDIT Statement

#### Syntax:

```

<archival audit statement> ::=
    ARCHIVALAUDIT = <logical value>. / <empty>

```

#### Semantics:

When ARCHIVALAUDIT = TRUE, the input queue is audited on the Archival Audit tapes. Recoverable input queues are also audited on the Archive tapes whenever ARCHIVALAUDIT = TRUE in the SYSTEM Section.

## AUDIT Statement

### Syntax:

<audit statement> ::= AUDIT = DUPLICATE./ <empty>

### Semantics:

The <audit statement> specifies the type of audit to be performed against this input queue.

When AUDIT = DUPLICATE, the Utility program creates duplicate control, input, and output files for additional recovery protection. During the running of the MCS, all messages through this queue and their corresponding outputs are written to the duplicate audit files, as well as to the primary audit files.

### Pragmatics:

All output associated with an input transaction is audited in duplicate. Such transactions are presented to the TP with MSG-DUPAUDIT (COMMON[16].[1:1]) set to 1. If a TP does not want certain output messages audited twice (even though they are in response to an input message which was audited in duplicate), it may designate this by resetting MSG-DUPAUDIT (COMMON[16].[1:1]) to 0. If this bit is so reset by the TP, and the TP wants subsequent output messages to be audited in duplicate for the same input transaction, then it must set the bit back to 1.

If duplicate audit files exist, a program can cause any output to be audited in duplicate by setting this bit to 1, even if the corresponding input message was not audited in duplicate.

When the MCS encounters a parity error while attempting to read an Input, Output, or Control record from the first (or primary) audit file, it then attempts to read the same record from the corresponding duplicate audit file. If successful, the MCS then rewrites this record back to the primary audit file.

While the same amount of disk space is used, the GEMCOS implementation of duplicate audit differs from the MCP's implementation in that only those I/Os selected by the user are duplicated.

In order to take full advantage of the parallel writes that are done to the duplicate audit files and to safeguard against the loss of both sets of a given audit file in the event of a spindle failure, the user may want to label-equate the duplicate files to packs other than those on which the primary audit files reside. If the user has more than one disk-pack controller, the two sets of queue files ideally should be split evenly across the controllers to prevent a controller malfunction from corrupting both sets of queue files.

When no inputqueues are declared in the TCL with AUDIT = DUPLICATE specified, only a single set of queue files is generated by the TCL compiler.

## AUDITINPUT Statement

Syntax:

```
<audit input statement> ::=  
    AUDITINPUT = <logical value>. / <empty>
```

Semantics:

The <audit input statement> specifies whether or not input to this queue will be audited in the input audit files. Setting AUDITINPUT = FALSE will preclude any recovery, including resubmission of transactions which cause a program to abort, for this input queue. Queues which have AUDITINPUT = FALSE may not specify RECOVERY = TRUE, ARCHIVALAUDIT = TRUE or AUDIT = DUPLICATE. The default value is TRUE.

## MEMORYLIMIT Statement

Syntax:

```
<memory limit statement> ::=  
    MEMORYLIMIT = <integer>. / <empty>
```

Semantics:

The <memory limit statement> is used to specify the number of messages allowed to queue up in memory before being tanked to disk. Its value must be between 1 and 255. Since only the last block of a message is placed in the memory queue, only one input block per message is affected by this statement. The default value for MEMORYLIMIT is 1.

Pragmatics:

Varying MEMORYLIMIT has a potentially high impact on disk I/O versus main-memory requirements and thus on throughput.

## MKE Statement

Syntax:

```
<message key statement> ::=  
    MKE = <message key list>. /  
    <message key statement> <message key statement>  
  
<message key list> ::=  
    <message key identifier> /  
    <message key list>, <message key identifier>  
  
<message key identifier> ::=  
    <mke> <user control statement> <module-function indices>
```



```

<mke> ::=
    <identifier> / <integer> / <string> /
    <hexadecimal code> <hex string>

<user control statement> ::= :USERCONTROL / <empty>

<module-function indices> ::=
    (<integer>, <integer>) / <empty>

```

#### Semantics:

The <message key statement> defines the message tag codes to be accepted by the program, as well as certain attributes associated with each message tag code.

When specified, the <user control statement> indicates that the message key is processed by the INPUTUSERHOOK procedure. (Refer to Section 4 for a discussion of this procedure.)

The <module-function indices> specify a pair of integers which accompany a message containing this message key to the COMMON area for the convenience of the user programmer. If the <module-function indices> are <empty>, zeros are returned to the COMMON area with messages containing this message key. Appendix A presents a layout of COMMON indicating the location of the module and function indices in the message passed to a User program.

#### Pragmatics:

If the message key is specified as an identifier, the first character must be non-numeric. If the message key is an integer, it must be all numeric. Message keys specified as either EBCDIC or hexadecimal strings must not begin with a space, and variable-length message keys with the same specification must use alphanumeric characters.

#### Example:

```

MKE = ABC123(123,1), ABC456(456,1).
MKE = ABX456.
MKE = 1234(678,23).
MKE = ABX457(1,2), 4"1AC1OD"(1,3).

```

## NOINPUT Statement

#### Syntax:

```

<no input statement> ::=
    NOINPUT = <logical value>. / <empty>

```

#### Semantics:

If the <no input statement> is set TRUE, this is a queue to be used by a NOINPUT program. No other <inputqueue statement> may be used, and only one <inputqueue section> may be specified for a program with NOINPUT = TRUE.

A NOINPUT program must be run by inputting a ?RUN Network Control command.

### QUEUEDEPTH Statement

#### Syntax:

```
<queue depth statement> ::=
    QUEUEDEPTH = <integer>. / <empty>
```

#### Semantics:

The <queue depth statement> specifies the number of messages which can back up in the queue before an additional copy of the program is executed.

If MAXCOPIES is greater than 1, and this statement is omitted, the queue depth is set to a high value, and no additional copies of the program are executed.

### RECOVERY Statement

#### Syntax:

```
<recovery statement> ::= RECOVERY = <logical value>. / <empty>
```

#### Semantics:

The <recovery statement> is used to indicate whether the program under which this Input queue is defined will participate in the data base/data communications synchronized recovery mechanism. If RECOVERY = FALSE, then this Input queue is recovered using the Checkpoint recovery mechanism.

If this statement is omitted, FALSE is assumed. A detailed discussion of recovery conventions is presented in section 5.

### TIMELIMIT Statement

#### Syntax:

```
<time limit statement> ::= TIMELIMIT = <integer>. / <empty>
```

#### Semantics:

The <time limit statement> specifies the length of time, in seconds, which should elapse between the point the queue depth rises above the limit specified by the <queue depth statement> and the point the MCS should invoke copies of this program. Copies are terminated based on INACTIVETIMEOUT.

When MAXCOPIES is greater than one and this statement is omitted, the time limit is set to a high value, and no extra copies of the program are executed.

### WAITFORAUDIT Statement

#### Syntax:

```
<wait for audit statement> ::=  
    WAITFORAUDIT = <logical value> . / <empty>
```

#### Semantics:

The <wait for audit statement> specifies whether the program for which this Input queue is defined, while processing messages from this queue, will delay issuing END TRANSACTION to the data-base manager until after the output primary message has been audited.

If WAITFORAUDIT = FALSE, it is possible, but unlikely, for duplicate processing of input to occur during recovery. It is also remotely possible for input messages to be reprocessed in a sequence different from the original sequence.

If the statement is omitted, FALSE is assumed. Recovery conventions are discussed in detail in section 5.

#### NOTE

WAITFORAUDIT = TRUE does not imply RECOVERY = TRUE.

## STATION SECTION

#### Syntax:

```
<station section> ::= <station define list>  
  
<station define list> ::=  
    <station define> /  
    <station define list> <station define>
```

```

<station define> ::=
    STATION <station name> (<station number>):
        <station description> /
    SWITCHSTATION <station name> (<station number>):
        <switch station description>

<station number> ::= <integer>

<station description> ::= <station statement list>

<station statement list> ::=
    <station statement> /
    <station statement list> <station statement>

<station statement> ::=
    <address statement> /
    <alternate stations statement> /
    <assign to program statement> /
    <broadcast statement> /
    <compute statement> /
    <continuous log-on statement> /
    <conversational statement> /
    <dial in statement> /
    <fixed length mkes statement> /
    <flush output statement> /
    <formatted blocked output statement> /
    <forms compose statement> /
    <host accesskey statement> /
    <identify statement> /
    <individual id statement> /
    <intercept statement> /
    <line analyzer statement> /
    <line statement> /
    <logical ack statement> /
    <may not be intercepted statement> /
    <maximum users statement> /
    <my use statement> /
    <no duplicates statement> /
    <no line statement> /
    <no queue statement> /
    <no title statement> /
    <one output per backup statement> /
    <portsize statement> /
    <portstation statement> /
    <program acknowledge statement> /
    <remarks statement> /
    <routing header statement> /
    <service messages statement> /
    <set MCS request set statement> /

```

```

<sign-on statement> /
<special statement> /
<station bit statement list> /
<station hostname statement> /
<station master statement> /
<station message key statement> /
<station system statement> /
<station your name statement> /
<system network control station statement>
<test statement> /
<transaction mode statement> /
<type statement> /
<user control statement> /
<user id mask statement> /
<valid users statement> /
<variable mke position statement> /
<variable user id position statement> /
<empty>

<switch station description> ::=
    <switch station statement list>

<switch station statement list> ::=
    <switch station statement> /
    <switch station statement list> <switch station statement>

<switch station statement> ::=
    <sign-on statement> / <valid users statement> / <empty>

```

#### Semantics:

The STATION section defines station functions and relations to the MCS.

A <station define> must contain a <station name> and a <station number>. In order for the MCS to associate the station declaration with a physical terminal, the station name should be one declared in the NDL. The station number may not be a duplicate of another <station number>. A <station number> may range from 1 to 200000.

A <switch station define> must contain a <station name> and a <station number>. This station may be switched between systems. A normal station must occur in one system (the system that the station would belong to initially). A SWITCHSTATION declaration must occur in the STATION section of each system to which it may be switched, and the station name and number must be the same in all declarations. Under the SWITCHSTATION, only SIGNON, VALIDUSERS, FIXEDMKE and FORMSCOMPOSE may be specified. The switch stations may appear in the appropriate DEVICE section within the systems the station can be switched between.

Example of STATION Define:

```
STATION SYSTEM/MONITOR (101):  
TEST = TRUE.  
SERVICEMESSAGES = TRUE.  
LINE = 0:0:1.  
ADDRESS = "AA".  
REMARKS = "YE OLD SYSTEM MONITOR ".
```

Example of SWITCHSTATION Define:

```
BEGIN SYSTEM SYS 1 (1):  
  ACCESSCONTROL = ACCESSKEY 84123 =  
    MKE1, MKE2.  
  •  
  •  
  •  
  STATION DEMO1 (1):  
    SIGNON = TRUE.  
    VALIDUSERS = ALL.  
  STATION DEMO2 (2):  
    SIGNON = TRUE.  
    VALIDUSERS = 84123.  
  •  
  •  
  •  
  DEVICE TD1 (1):  
    STALIST = DEMO1, DEMO2.  
END  
BEGIN SYSTEM SYS2 (2):  
  ACCESSCONTROL = ACCESSKEY 85100 = MKE3, MKE4.  
  •  
  •  
  •  
  STATION DEMO3 (3)  
    SIGNON = TRUE.  
    VALIDUSERS = ALL.  
  SWITCHSTATION DEMO2 (2):  
    SIGNON = TRUE.  
    VALIDUSERS = 85100.  
  •  
  •  
  •  
  DEVICE TD2 (2):  
    STALIST = DEMO3, DEMO2.  
END
```

## ADDRESS Statement

Syntax:

`<address statement> ::= ADDRESS = <address part>.`

`<address part> ::= <address> / <address pair> / NULL`

`<address> ::= <string>`

`<address pair> ::= (<address>, <address>)`

Semantics:

The `<address statement>` specifies the poll and select address characters for the station. The station address is not required and is for documentation only.

Example:

`ADDRESS = "AB".`

## ALTERNATE STATIONS Statement

Syntax:

`<alternate stations statement> ::=`  
    `ALTERNATE =`  
    `<alternate station list>. /`  
    `<empty>`

`<alternate station list> ::=`  
    `<alternate station> /`  
    `<alternate station list>, <alternate station>`

`<alternate station> ::= <station name>`

#### Semantics:

The <alternate stations statement> specifies an alternate which may be used for output when the specified station is marked as inoperative. If several alternates are specified, each is checked for being operative with the first specified as having the highest priority. If all alternates are disabled, the alternates to the alternates are also checked, down to any level until all possibilities are exhausted.

Output to an alternate of a station on a line will be invoked only when the station is specifically disabled by a system control message. Output to an alternate of a station having no line (NO-LINE = TRUE) will be invoked only when the station is enabled.

#### Example:

```
ALTERNATE = CMPT, SSVR.
```

### ASSIGNTOPROGRAM Statement

#### Syntax:

```
<assign to program statement> ::=  
    ASSIGNTOPROGRAM = <program name>. / <empty>
```

#### Semantics:

The <assign to program statement> allows a station to assign itself permanently to a program. In this case, all messages are routed directly to and from the program without having to enter message keys at the station.

### BROADCAST Statement

#### Syntax:

```
<broadcast statement> ::=  
    BROADCAST = <logical value>. / <empty>
```

#### Semantics:

The <broadcast statement> specifies that the station has been physically optioned to receive messages under a broadcast select. If it is not specified, FALSE is assumed.



### Pragmatics:

If a message is to be sent to a group of stations (area), and the group includes all stations on a line, all of which have `BROADCAST = TRUE`, then the MCS issues a special broadcast select write for the line. All stations on the line which have been physically optioned to receive messages under a broadcast select will receive the message, including stations which are not attached to GEMCOS.

### Example:

```
BROADCAST = TRUE.
```

## COMPUTE Statement

### Syntax:

```
<compute statement> ::= COMPUTE = <logical value>. / <empty>
```

### Semantics:

The <compute statement> specifies if this station is allowed to use the compute function of the CONTROL process. This statement need not be specified unless to override the <master compute statement> specified in the GLOBAL section. Detailed information on the use of the Compute function is presented in section 8.

### Example:

```
COMPUTE = TRUE.
```

## CONTINUOUSLOGON Statement

### Syntax:

```
<continuous log-on statement> ::=  
CONTINUOUSLOGON = <logical value>. / <empty>
```

### Semantics:

The <continuous log-on statement> specifies whether the MCS should "remember" who was logged on to the station following a termination of the network (either normally or abnormally). Additionally, if the station's station bits had been changed during a valid log-on or log off, the <continuous log-on statement> specifies that the current value of the station bits should also be "remembered." If the statement is omitted or FALSE, all users logged on to the station when the MCS terminates are automatically logged off when the MCS is started again.

## CONVERSATIONAL Statement

### Syntax:

```
<conversational statement> ::=  
    CONVERSATIONAL = <logical value>. / <empty>
```

### Semantics:

The <conversational statement> specifies that the station may enter into a dialogue with a program. When this statement is set to TRUE, a Conversational row is assigned to the station in anticipation of in-core storage of a conversation. If conversations are never stored in core, the row is never accessed. Deallocation of the row is under control of the user TP. The size of the row depends on the size of the Conversational area of the particular program in conversation with the station. See Section 4 for a more complete description.

## DIALIN Statement

### Syntax:

```
<dial-in statement> ::= DIALIN = <logical value>. / <empty>
```

### Semantics:

The <dial-in statement> indicates those stations which are specified in NDL to be dial-in stations. This statement in the TCL is for documentation only. Refer to the IDENTIFY statement and the NOLINE statement for further details of dial-in capabilities.

## FIXEDLENGTHMKE Statement

### Syntax:

```
<fixed length mkes statement> ::=  
    FIXEDMKE = <integer>. / <empty>
```

### Semantics:

The <fixed length mkes statement> indicates that this station handles only fixed-length message keys. All message keys entered through this station must be <integer> characters long. FIXED-MKE may take on a value between 1 and 15. For a fixed-length message key station, it is assumed that the first <integer> bytes (after skipping leading blanks) of the message starting at MKEPOSITION (refer to the <variable mke position statement> at the end of the STATION section) constitute the message key. For variable length message-key stations, it is assumed that the first nonalphanumeric character (after skipping leading blanks) terminates the message key.

## FLUSHOUTPUT Statement

Syntax:

```
<flush output statement> ::=  
    FLUSHOUTPUT = <logical value>. / <empty>
```

Semantics:

The <flush output statement> specifies that this station's Output Message queue be purged after a synchronized recovery.

## FORMATTEDBLOCKEDOUTPUT Statement

Syntax:

```
<formatted blocked output statement> ::=  
    FORMATTEDBLOCKEDOUTPUT = <logical value>. / <empty>
```

Semantics:

The <formatted blocked output statement> specifies if the MCS is to block output to this buffered terminal even though it has been formatted. This option is intended for large formatted output messages to hard-copy devices when users want to avoid the necessity of terminal operator intervention to request subsequent pages of output.

If the statement is FALSE or omitted, formatted output is transmitted unblocked.

## FORMSCOMPOSE Statement

Syntax:

```
<forms compose statement> ::=  
    FORMSCOMPOSE = <message-id>. / <empty>
```

Semantics:

The <forms compose statement> allows a user to specify a format that is delivered to a station when it is enabled. In this manner tab settings and special messages may be delivered to specific stations when they are enabled. In the DEVICE section where the station is specified, a format must be related to the given message-ID using the FORMATSOUT syntax.

## HOST ACCESSKEY Statement

Syntax:

```
<host accesskey statement> ::=  
    HOSTACCESSKEY = <identifier>. / <empty>
```

Semantics:

The <identifier> specified in the <host accesskey statement> is placed in the accesskey field in the routeheader of a TYPE = ROUTEHEADER station and is used for security checking. Refer to section 13 for a more complete discussion. The default value is blanks.

Example:

```
HOSTACCESSKEY = HACKER.
```

## IDENTIFY Statement

Syntax:

```
<identify statement> ::=  
    IDENTIFY = <logical value>. / <empty>
```

Semantics:

The <identify statement> specifies those stations on a dial-in line which are required to identify themselves before being enabled at initial connect time.

A TCL station name should be entered on the terminal in response to the IDENTITY REQUIRED prompt. This station name should not exist in the NDL source deck (i.e., it should be declared in the TCL as NOLINE = TRUE). To acknowledge a successful identification, IDENTITY ACCEPTED is sent to the terminal. The terminal then assumes the TCL station attributes of this station declaration in the TCL.

If IDENTIFY = FALSE for a terminal, it assumes the characteristics of the corresponding TCL declaration.

For each unsuccessful attempt, an error message is sent to the network monitor stations, and another prompt is sent to the terminal.

If all stations on a line are IDENTIFY = FALSE, they are all enabled at initial connect time. If there is at least one IDENTIFY = TRUE station on the line, then the IDENTIFY = FALSE stations are enabled when one of the IDENTIFY = TRUE stations identifies itself.

When the line is disconnected, stations revert to their original state and require identification.

## INDIVIDUALID Statement

Syntax:

```
<individualid statement> ::=  
    INDIVIDUALID = <logical value>. / <empty>
```

Semantics:

The <individualid statement> specifies if the station is to have a special individual identifier associated with each user who is logged on to the station. When TRUE, the MCS accepts a 2-word identifier from the Access Control module with each valid log-on request. This 2-word identifier is passed to the application program with each transaction entered by the user. Access Control is discussed in detail in section 7.

If the statement is omitted, FALSE is assumed. This feature is not compatible with the CONTINUOUSLOGON feature and should not be specified TRUE on a station for which CONTINUOUSLOGON = TRUE.

## INTERCEPT Statement

Syntax:

```
<intercept statement> ::=  
    INTERCEPT = <logical value>. / <empty>
```

Semantics:

The <intercept statement> specifies whether the station can intercept input and/or output messages from another station in the network. All stations can be intercepted except those which are defined as MAYNOTBEINTERCEPTED = TRUE. Interception is started and stopped by a control message issued by an intercept station. Interception is transparent to the intercepted station. If the <intercept statement> is omitted, FALSE is assumed.

Example:

```
INTERCEPT = TRUE.
```

## LINE Statement

Syntax:

```
<line statement> ::= LINE = <dcp address>. / <empty>
```

```
<dcp address> ::= <dcp> : <cluster> : <adapter>
```

```
<dcp> ::= <integer>
```

<cluster> ::= <integer>

<adapter> ::= <integer>

Semantics:

The <line statement> defines the port where the station is attached. <dcp> ranges from 0 to 7. <cluster> and <adaptor> range from 0 to 15. This statement is for documentation only.

Example:

LINE = 0:01:3.

## LINEANALYZER Statement

Syntax:

<line analyzer statement> ::=  
LINEANALYZER = <logical value>. / <empty>

Semantics:

The <line analyzer statement> specifies whether this station is a line monitor using a DLM adapter and associated DCP requests. If true, then all input from the specified station is passed directly to an analyzer section of the MCS for analysis and printer output. If the statement is not specified, FALSE is assumed.

Example:

LINEANALYZER = TRUE.

## LOGICALACK Statement

Syntax:

<logical ack statement> ::=  
LOGICALACK = <logical value>. / <empty>

Semantics:

The <logical ack statement> specifies how the NDL LOGICALACK option should be set for this station. When LOGICALACK = TRUE, input messages from this station are not acknowledged until after the message is audited by the MCS. If PROGRAMACK is also TRUE, the acknowledgement is delayed until after the receiving program indicates the message is processed. For a User program, this is done when the program indicates it is finished with the transaction (pass of control of 3). Process programs are informed when PROGRAMACK is TRUE, and they may specifically request that the MCS acknowledge the station by means of a special SEND-MESSAGE call (refer to appendix C). If a Process program does not specifically request the stations to be acknowledged, the MCS acknowledges after the transaction is complete.

If the <logical ack statement> is omitted, FALSE is assumed.

## MAYNOTBEINTERCEPTED Statement

### Syntax:

```
<may not be intercepted statement> ::=  
    MAYNOTBEINTERCEPTED = <logical value>. / <empty>
```

### Semantics:

The <may not be intercepted statement> specifies if the station can be intercepted by an INTERCEPT = TRUE station. Value TRUE indicates that it cannot be intercepted. If the statement is not specified, FALSE is assumed.

### Example:

```
MAYNOTBEINTERCEPTED = TRUE.
```

## MAXUSERS Statement

### Syntax:

```
<maximum users statement> ::=  
    MAXUSERS = <integer>. / <empty>
```

### Semantics:

The <maximum users statement> specifies the maximum number of operators or users which can be concurrently signed on to the specified station. If the statement is omitted, one (1) is assumed. If the maximum number of users is greater than 1, then each user signing on to the station must precede any input, including the log-on, with a 1-character identifier. This 1-character identifier is removed from the message which is presented to the TP but is passed to the TP in the control section of its Common area row. The layout of COMMON, indicating the location of the identifier in the message, is presented in appendix A.

## MYUSE Statement

### Syntax:

```
<my use statement> ::= MYUSE = <my use specifier>. / <empty>
```

```
<my use specifier> ::=  
    <input only my use> / <output only my use> /  
    <input - output my use>
```

```
<input only my use> ::= IN/1
```

```
<output only my use> ::= OUT/2
```

```
<input - output my use> ::= IO/3
```

Semantics:

The <my use statement> specifies the I/O capability of the station. If it is not specified, the NDL MYUSE is used.

Example:

```
MYUSE = IO.  
MYUSE = 2.
```

## NODUPLICATES Statement

Syntax:

```
<no duplicates statement> ::=  
    NODUPLICATES = <logical value>. / <empty>
```

Semantics:

The <no duplicates statement> specifies that the MCS is to hold back possible duplicate messages during recovery. If it is not specified, FALSE is assumed, and possible duplicate messages are released during recovery. A detailed discussion on output message analysis is presented in section 5. Utilization of this option is not normally recommended except in instances where retransmission of the last output delivered to a station could produce unfavorable results.

## NOLINE Statement

Syntax:

```
<no line statement> ::= NOLINE = <logical value>. / <empty>
```

Semantics:

The <no line statement> specifies that this station has no equivalent station defined in NDL. This station can be attached to a dial-in station and can have messages directed to it by an application program. Alternate stations can be specified for this station so that output can be processed from its queue.

## NOQUEUE Statement

Syntax:

```
<no queue statement> ::=  
    NOQUEUE = <logical value>. / <empty>
```



#### Semantics:

The <no queue statement> specifies the station's ability to queue output from the data communications system. It is set to TRUE if only one output message can be sent for each input to the system. This parameter would be used to accommodate another manufacturer's system which could not accept more than one response for each input. If the statement is not specified, FALSE is assumed.

#### Example:

```
NOQUEUE = TRUE.
```

### NOTITLE Statement

#### Syntax:

```
<no title statement> ::=
    NOTITLE = <logical value>. / <empty>
```

#### Semantics:

The <no title statement> specifies whether the station's output messages should start with the date-time title. If not specified, FALSE is assumed, and the title is attached to the output message. The title is always built and forwarded to the output audit unless the program generating the output sets the NOTITLE flag in COMMON when passing control to the MCS.

#### Example:

```
NOTITLE = TRUE.
NOTITLE = FALSE.
```

### ONEOUTPUTPERBACKUP Statement

#### Syntax:

```
<one output per backup statement> ::=
    ONEOUTPUTPERBACKUP = <logical value>. / <empty>
```

#### Semantics:

The <one output per backup statement> indicates whether the next output to a station should be held up until the audit file receives acknowledgement that the previous output has been delivered. If the statement is FALSE or <empty>, no output is held up. If TRUE, then the maximum number of possible duplicate messages at this station after recovery is reduced to one.

## PORTSIZE Statement

### Syntax:

`<portsize statement> ::= PORTSIZE = <integer>. / <empty>`

### Semantics:

The `<portsize statement>` only has meaning for a station with `PORTSTATION = TRUE`. The MCS and the utility must both be compiled with the `PORTS $` option set. It specifies the maximum number of characters that GEMCOS will read from or write to the port associated with this station. Messages received by GEMCOS from a port station will be truncated at the largest value of all `PORTSIZE`s for all portstations. Messages sent by GEMCOS to a port station will be sent in pieces of its `PORTSIZE` length. The default value of `PORTSIZE` is 2000 and the maximum value is 3000.

## PORTSTATION Statement

### Syntax:

`<portstation statement> ::= <logical value>. / <empty>`

### Semantics:

When `PORTSTATION` is set true, GEMCOS attempts to receive input and output via a `PORT` file rather than a datacom station. If the station is a `TYPE = ROUTEHEADER` station then GEMCOS may communicate with a GEMCOS on another host in a BNA network. Alternately, the user may write a program which uses a `PORT` file to simulate a GEMCOS station. File attributes of `PORT` used by GEMCOS may be specified in the TCL by the use of the `MYNAME` statement in the global section, or the `STATIONHOSTNAME`, `STATIONYOURNAME` and `PORTSIZE` statements in the station section. The internal name of the `PORT` file used by GEMCOS is `HOSTPORT` and the default title is `GEMPORT`. The MCS and the utility must both be compiled with the `PORTS $` option set. The default value of `PORTSTATION` is false.

Examples:

1. This example allows a program on the host named OPOTIKI to open a port and act like a GEMCOS station.

```
STATION ITSAFORT (1):  
  PORTSTATION = TRUE.  
  STATIONHOSTNAME = OPOTIKI.  
  NOTITLE = TRUE.
```

2. This example shows the TCLs required to allow a GEMCOS on Host Whakatane to communicate with GEMCOS on Host Taneatua.

Examples:

1. This example allows a program on the host named OPOTIKI to open a port and act like a GEMCOS station.

```
STATION ITSAFORT (1):  
  PORTSTATION      = TRUE.  
  STATIONHOSTNAME  = OPOTIKI.  
  NOTITLE          = TRUE.
```

2. This example shows the TCLs required to allow a GEMCOS on Host Whakatane to communicate with GEMCOS on Host Taneatua.

```
% TCL on Host Taneatua
```

```
STATION WHAKATANE (2):  
  % WHAKATANE IS HOSTNAME BY DEFAULT  
  PORTSTATION      = TRUE.  
  TYPE             = ROUTEHEADER.  
  STATIONYOURNAME  = OHOPE.  
  NOTITLE          = TRUE.  
  MKE              = WHAKAI,WHAKA2.
```

```
% TCL on Host Whakatane
```

```
MYNAME = OHOPE.  
% MATCHES STATIONYOURNAME ABOVE
```

.

.

```
STATION X(1):  
  PORTSTATION      = TRUE.  
  TYPE             = ROUTEHEADER.  
  STATIONHOSTNAME  = TANEATUA.  
  NOTITLE          = TRUE.  
  MKE              = TANE1,TANE2.
```

## PROGRAMACK Statement

Syntax:

```
<program acknowledge statement> ::=  
    PROGRAMACK = <logical value>. / <empty>
```

Semantics:

The <program acknowledge statement> is used in conjunction with the NDL LOGICALACK statement and specifies whether a program should acknowledge an input message. If specified TRUE, and NDL LOGICALACK is TRUE, messages are acknowledged only when the program indicates they should be. Refer to the discussion for the <logical ack statement>.

If the statement is omitted, FALSE is assumed.

Example:

```
PROGRAMACK = TRUE.
```

## REMARKS Statement

Syntax:

```
<remarks statement> ::=  
    REMARKS = <remarks string>. / <empty>  
  
<remarks string> ::= <string> / <remarks string>, <string>
```

Semantics:

The <remarks statement> specifies a <string> that is to be saved for documentation purposes. CONTROL = LIST is used to print out this listing. The maximum length of a <remarks string> is 100 characters. The <remarks statement> is not returned.

## ROUTEHEADER Statement

Syntax:

```
<routing header statement> ::=  
    ROUTEHEADER = <logical value>. / <empty>
```

#### Semantics:

The <routing header statement> specifies whether the station begins each message with a routing header and expects each message directed to it to be preceded by a routing header. This implies that the station is a computer or message concentrator which services another set of stations. The routing header is transmitted with the message throughout the data communications system and returned to the station preceding the output message. If the statement is not specified, FALSE is assumed. An alternative type of routeheader may be made by specifying TYPE = ROUTEHEADER. See section 13, computer-to-computer communication for a more complete discussion.

### SERVICEMESSAGES Statement

#### Syntax:

```
<service messages statement> ::=  
    SERVICEMESSAGES = <logical value> ./ <empty>
```

#### Semantics:

The <service messages statement> specifies whether the station is to receive service messages. If not specified, FALSE is assumed.

#### Example:

```
SERVICEMESSAGES = TRUE.  
SERVICEMESSAGES = FALSE.
```

### SETMCSREQUESTSET Statement

#### Syntax:

```
<set MCS request set statement> ::=  
    SETMCSREQUESTSET = <logical value> ./ <empty>
```

#### Semantics:

The <set MCS request set statement> allows a station to change NDL request sets when switching from some other MCS to GEMCOS and back again. GEMCOS assumes that any station for which SETMCSREQUESTSET = TRUE is specified uses NDL request set number 5. The request set number for the station is changed to 5 whenever the station is released to GEMCOS and is then changed to 1 whenever the station is released to some other MCS.

If the statement is omitted or FALSE, and the station is assigned to GEMCOS in the NDL, then request set 1 is assumed. If it is omitted or FALSE and the station is not assigned to GEMCOS in the NDL, the request set is not changed when the station is released from GEMCOS.

## SIGNON Statement

### Syntax:

<sign-on statement> ::= SIGNON = <logical value>. / <empty>

### Semantics:

The <sign-on statement> is used to indicate whether users must sign on to this station prior to entering transactions. If this statement is omitted, FALSE is assumed.

If SIGNON = TRUE, the operator at the station must sign on as one of the valid users listed in the <valid users statement>. If VALIDUSERS = ALL, the station operator must sign on as one of the users listed in the <access control statement> of the <system section>.

## SPECIAL Statement

### Syntax:

<special statement> ::= SPECIAL = <logical value>./ <empty>

### Semantics:

The <special statement> specifies whether the station is assumed to be part of the special group of stations. A special station is "enabled" immediately after a communication error occurs if the station is not marked "down" and the error was not a line error. Special stations also may put stations back in service. (Refer to \$ INPUT MESSAGE in section 8.) If this statement is not specified, FALSE is assumed.

### Example:

SPECIAL = TRUE.

## STATIONBIT Statement

### Syntax:

<station bit statement list> ::=  
    <station bit statement> /  
    <station bit statement list> <station bit statement> /  
    <empty>

<station bit statement> ::= STATIONBIT <sta-bit specifier>.

<sta-bit specifier> ::=  
    <individual sta-bit specifier> /  
    <field sta-bit specifier>

<field sta-bit specifier> ::=  
    (<bit number>:<field length>) = <sta-bit value>

<sta-bit value> ::= <integer> / <hex string>

<field length> ::= [a number from 1 to 48]

<bit number> ::= [a number from 0 to 47]

#### Semantics:

Associated with each station are up to 48 bits of logical data. These bits are passed to application programs along with the input message to be processed. The <station bit statement list> specifies the value of these bits. The <bit number>s are the same as when they are passed to the application program. Any bit not specified is assumed FALSE.

The station bits may be assigned a value either individually or in groups. If the bits are to be assigned values in groups, the first <bit number> is the starting bit of the field, and the <field length> is the length of the field. If the <sta-bit value> is an integer, it must be small enough to fit in the field specified. If the <sta-bit value> is a hexadecimal string, it must contain an even number of digits; the bits are extracted from the string from left to right.

The station bits may be altered by an Access Control program during a valid sign-on or sign-off verification (refer to section 7).

#### Example:

```
STATIONBIT(0) = TRUE.  
STATIONBIT(10) = TRUE.  
STATIONBIT(13:14) = 4097.  
STATIONBIT(30:15) = 4"7FFC".
```

## STAMASTER Statement

#### Syntax:

<station master statement> ::=  
    STAMASTER = <logical value> . / <empty>

#### Semantics:

The <station master statement> specifies whether the station is to be considered master. Normally, a station is a slave to the data communications system; but in some cases, if the station is a computer, the station is in control of the message flow. This information is passed to the Data Communications Controller (DCC) which could use it to relieve line contention in computer-to-computer communication. If the statement is omitted, FALSE is assumed.

#### Example:

```
STAMASTER = TRUE.  
STAMASTER = FALSE.
```

## STASYSTEM Statement

Syntax:

```
<station system statement> ::=  
    STASYSTEM = <string>. / <empty>
```

Semantics:

The <station system statement> specifies a system name to be used in the message title if a message is switched from this station. If the statement is not specified, then the system name is used.

Example:

```
STASYSTEM = "MYSYSTEM".
```

## STATIONHOSTNAME Statement

Syntax:

```
<station host name statement> ::=  
    STATIONHOSTNAME = <identifier>. / <empty>
```

Semantics:

The <station host name statement> only has meaning for a station with PORTSTATION = TRUE. The HOSTNAME attribute of the PORT SUBFILE used by GEMCOS for this station is set to <identifier>. The MCS and the utility must both be compiled with the PORTS \$ option set. The default value of STATIONHOSTNAME is the station name.

## STATIONMESSAGEKEY Statement

Syntax:

```
<station message key statement> ::=  
    MKE = <station message key list>.  
  
<station message key list> ::=  
    <mke>/<mke>, <station message key list>  
  
<mke> ::=  
    <identifier> / <integer> / <string> /  
    <hexadecimal code> <hexadecimal string>
```



#### Semantics:

The <station message key statement> provides a means of station-to-station communication. A message entered from station A, which contains a message key declared in the TCL for station B will be routed directly to station B. If station B is a ROUTEHEADER = TRUE station, a routeheader will be added and any reply from station B will be directed back to station A. Programs may not route to a station by message key.

#### Example:

```
MKE = AB2, 123, "20X", 4"C1C2F3".
```

### STATIONTIMEOUT Statement

#### Syntax:

```
<station time out statement> ::=  
    STATIONTIMEOUT = <logical value>./ <empty>
```

#### Semantics:

The <station timeout statement> specifies whether the station should be disabled by timeout errors. If TRUE is specified, and no good results are received from the station within six minutes of a transmission to the station, the station will timeout. If FALSE is specified, the station will not time out. This statement is similarly used by PAGER to determine whether a station should automatically be taken out of paging mode after ten minutes with no paging activity. If this statement is omitted, FALSE is assumed.

### STATIONYOURNAME Statement

#### Syntax:

```
<station your name statement> ::=  
    STATIONYOURNAME = <identifier>./ <empty>
```

#### Semantics:

The <station your name statement> only has meaning for a station with PORTSTATION = TRUE. The YOURNAME attribute of the PORT SUBFILE used by GEMCOS to this station is set to <identifier>. The MCS and the utility must both be compiled with the PORTS \$ option set. The default value of STATIONYOURNAME is null.

### SYSTEMNETWORKCONTROLSTATION Statement

#### Syntax:

```
<system network control station statement> ::=  
    SYSTEMSPOSTATION = <logical value>./ <empty>
```

Semantics:

The <system network control station statement> specifies whether the station can send system control messages. The system sets this attribute to TRUE for any stations named in the <system network control statement> in the GLOBAL section. If this statement is not specified, FALSE is assumed.

## TEST Statement

Syntax:

<test statement> ::= TEST = <logical value> . / <empty>

Semantics:

The <station test statement> specifies whether the station is part of a test group of stations. If this statement is not specified, FALSE is assumed. These stations may be enabled or disabled as a group using the "ENABLE TEST" or "DISABLE TEST" Network Control commands. They could be used to test a portion of the physical system before enabling the entire system.

Example:

```
TEST = TRUE.  
TEST = FALSE.
```

## TRANSACTIONMODE Statement

Syntax:

<transaction mode statement> ::=  
TRANSACTIONMODE = <logical value> ./ <empty>

Semantics:

The <transaction mode statement> specifies whether a station is allowed to transmit a new input before receiving the response for the previous input transaction. If TRUE, the MCS returns the error response BUSY for any input from the station prior to transmitting a response to the current transaction for the station.

The BUSY response may not reach the station as the next event following the discarding of the transaction in error. The valid response from a TP to the initial input, or queued secondary messages generated to the station as the result of other transactions on the network, can be transmitted before the BUSY notification is delivered.

If this statement is omitted, FALSE is assumed, and the MCS accepts input from the station at the operator's direction.

## TYPE Statement

Syntax:

```
<type statement> ::=
    TYPE = <type value> ./ <empty>

<type value> ::=
    STANDARD / AP300 / MT600 / MT700 / ROUTEHEADER
```

Semantics:

The <type statement> is used to specify to GEMCOS whether this station requires special handling. The meanings of the <type value>s are as follows:

1. STANDARD specifies no special handling is required. This is the default value.
2. AP300 specifies that the station is to be treated as a Burroughs AP300 printing terminal. Any messages received from this station are assumed to be status messages and are sent to the program in its system which has been defined to be AP300STATUS = TRUE. If no such program exists, the status is stored in the station table and is printed as part of a ?STATUS system control message.
3. MT600 and MT700 <type value>s are synonymous. They specify that the station is to be treated as one of the Burroughs MT600 or MT700 family of terminals. Any input message containing a valid header ("DC4 <char> DC1") undergoes the following transformation:
  - a. The header is stripped from the message.
  - b. The MSG-MTCHAR field (COMMON[19].[47:8]) and CONTROLWORDS[8].[43:8] are set to <char>.
  - c. If there is a trailer on the message ("DC4 E DC1"), it is stripped from the message and the MSG-MTDEL field (COMMON[19].[39:1]) and CONTROLWORDS[8].[44:1] are set to 1.
  - d. When a message is sent from a program to a station of TYPE = MT600 or MT700, the COMMON AREA fields are checked. If the MSG-MTCHAR field or CONTROLWORDS[8].[43:8] is zero, then no header is put on the message. If the MSG-MTCHAR field or CONTROLWORDS[8].[43:8] contains a character, then it is used in the header and put on the message. A trailer is appended if MSG-MTDEL or CONTROLWORDS[8].[44:1] is equal to 1.
4. ROUTEHEADER specifies that a standard GEMCOS ROUTEHEADER will be added to every message sent to this station and will be expected on every message received from this station. It is used in computer-to-computer communication between GEMCOS MCS's on different Burroughs systems. See section 13 Computer to Computer Communication for a more complete discussion.

## USERCONTROL Statement

Syntax:

```
<user control statement> ::=  
    USERCONTROL = <logical value> ./ <empty>
```

Semantics:

When the <user control statement> is set to TRUE for a station, all transactions entered at that station will be processed by the INPUTUSERHOOK procedure. (Refer to section 4 for a discussion of this procedure.)

## USERIDMASK Statement

Syntax:

```
<user id mask statement> ::=  
    USERIDMASK = <integer > . / <empty>
```

Semantics:

When a multi-user station is in use, the <user id mask statement> is ANDed with the user-ID entered at log-on by the operator. This forms the internal user-ID associated with that log-on. Thus, depending on the mask that is selected, only certain bits of the user-ID character would be used.

The default value for the USERIDMASK is 255. This means that all bits of the user-ID are used.

## VALIDUSERS Statement

Syntax:

```
<valid users statement> ::=  
    VALIDUSERS = <accesskey list> . / <empty>
```

```
<accesskey list> ::=  
    <access code> / <access code>, <access key list> / ALL
```

#### Semantics:

A list of valid user codes can be declared to enforce user code validation at sign-on time. ALL indicates any user can sign on. If the statement is omitted and <signon> is TRUE, ALL is assumed. This statement has no meaning if <sign-on> is FALSE.

#### Example:

```
VALIDUSERS = ALL.  
VALIDUSERS = 84080,84090, HATCH.
```

### VARIABLEMKEPOSITION Statement

#### Syntax:

```
<variable mke position statement> ::=  
    MKEPOSITION = <integer>. / <empty>
```

#### Semantics:

The <variable mke position statement> allows the user to specify, on a station basis, at what position in the message the message key will appear.

The default value is 0, which has one of the following effects.

1. For a single user station, the MKE is expected to start at the first token in the message.
2. For a multi-user station with no user-ID position specified, the MKE is expected to start at the second token in the message (preceded by the user-ID).
3. For a multi-user station with a user-ID position specified, the MKE is expected to start at the first token in the message. Care must be taken in this case if standard GEMCOS access control is used.

### VARIABLEUSERIDPOSITION Statement

#### Syntax:

```
<variable user id position statement> ::=  
    USERIDPOSITION = <integer>. / <empty>
```

#### Semantics:

The <variable user id position statement> allows the user to specify at what position in the message the user-ID of a multi-user station will appear.

Default value for the USERIDPOSITION is 0. This value indicates that the user-ID is the first token

## DEVICE SECTION

Syntax:

```
<device section> ::= <device define list> / <empty>

<device define list> ::=
    <device define> /
    <device define list> <device define>

<device define> ::=
    DEVICE = <device name>
    (<device number>): <device description>

<device name> ::= <identifier>

<device number> ::= <integer>

<device description> ::=
    <station list statement> <device statement list>

<device statement list> ::=
    <device statement> /
    <device statement list> <device statement> / <empty>

<device statement> ::=
    <input formats statement> /
    <output formats statement> /
    <nonformatted mkcs statement>
```

Semantics:

The <device section> is used to group stations by device class and to provide information necessary for formatting input and output messages. It is in this section that the intersection of device class and message key or message-ID is identified for use in locating the proper format to be applied to a message.

Example:

```
DEVICE TD800 (7):
    STALIST = ABC123, TCXY2.
    FORMATSIN:
        ABFMT = INQ, TC4(10), 1236(20).
    FORMATSOUT:
        OUTFMT = 672, XY17, TD4.
    NONFORMATTEDMKES = MK1(10), MK2(20).
```

## FORMATSIN Statement

Syntax:

```
<input formats statement> ::=  
    FORMATSIN: <input formats list>  
  
<input formats list> ::=  
    <input format association> /  
    <input formats list> <input format association>  
  
<input format association> ::= <format-id> = <mke list>.  
  
<format-id> ::= <identifier>  
  
<mke list> ::=  
    <mke specification> / <mke list>, <mke specification>  
  
<mke specification> ::= <mke> <item count>  
  
<item count> ::= (<integer>) / <empty>
```

Semantics:

The <input formats statement> specifies which format to apply to an input message from a station in this device class when it contains one of the listed message keys.

The MCS determines if a message is to be formatted by searching a table. The table coordinates are message key and device class. This statement, therefore, is used to determine the message key coordinates for each format which applies to this device class.

Also associated with a message key and a device class may be an <item count>. If specified, this number is passed to the TP through the control area of the TP's Common row. The layout of COMMON, indicating the location of the item count in the message passed to a TP, is presented in appendix A. The <item count> has no meaning to the MCS, and its utilization is user defined.

## FORMATSOUT Statement

Syntax:

```
<output formats statement> ::=  
    FORMATSOUT: <output formats list>  
  
<output formats list> ::=  
    <output format association> /  
    <output formats list> <output format association>
```

<output format association> ::= <format-id> = <msg-id list>.

<msg-id list> ::= <message-id> / <msg-id list>, <message-id>

<message-id> ::= <identifier>

Semantics:

The <output formats statement> specifies information similar to the information specified in the <input formats statement>, only with regard to output messages. The <message-id> is the identifier placed in word 9 of the Common area by the User program with an output message to be formatted (see Appendix A). Each <format-id> must be six characters or less. Note carefully the difference between message keys (which are used as routing criteria for input messages and which help determine input formatting) and output message-ids (which are used for determining output formatting).

## NONFORMATTEDMKES Statement

Syntax:

<nonformatted mkes statement> ::=  
NONFORMATTEDMKES = <mke list>.

Semantics:

The <nonformatted mkes statement> specifies nonformatted message keys which have an <item count> associated with them. This <item count> is passed on to the TP associated with this MKE when it is entered through a station of this device class. This statement is optional and need only be used for message keys which do not undergo input formatting in this device but which have an item count associated with them.

## STALIST Statement

Syntax:

<station list statement> ::= STALIST = <station name list>.

Semantics:

The <station list statement> specifies all the stations to be considered as part of this device class.

## AREA STATION SECTION

Syntax:

<area section> ::= <area list> / <empty>



<area list> ::= <area define> / <area define> <area list>

<area define> ::=  
    AREA <area type> <area name>  
    (<area number>): <area description>

<area type> ::= BROADCAST / ROTARY/ <empty>

<area name> ::= <identifier>

<area number> ::= <integer>

<area description> ::= <area statement list>

<area statement list> ::=  
    <area statement> /  
    <area statement list>, <area statement>

<area statement> ::= <area station list statement>

#### Semantics:

The AREA section allows for the logical grouping of stations. This section is not required, but it could supply valuable information to the data communications system.

An <area define> must contain an <area name> and <area number>. The <area name> and <area number> may not be a duplicate of any other <area define>. An <area number> may range from 1 to 89.

If <area type> is BROADCAST, a message addressed to <area number> is delivered to all stations defined within that area. If <area type> is ROTARY, a message addressed to <area number> is delivered to the station defined in the area which has the smallest number of messages queued for it at that time. If <area type> is empty, BROADCAST is assumed.

## AREA STALIST Statement

#### Syntax:

<area station list statement> ::=  
    STALIST = <area station name list>.

<area station name list> ::=  
    <area station list> / <all station designator>

<all station designator> ::= = <slash> = / ALL

```

<area station list> ::=
    <area station name> /
    <area station list>, <area station list>

<area station name> ::=
    <station name> / NOT <station name> /
    <group station name> / <area name>

<group station name> ::=
    <partial station name> <slash> =

<partial station name> ::= <generalized identifier>

```

#### Semantics:

The required <area station list statement> specifies the stations or subareas that are to be considered in the area. The NOT <station name> <area station name> is used to exclude stations and should follow an <all station designator> or <group station name>.

If an <area name> is used in the STALIST, this area is treated as a subarea and must be declared in a previous <area define>. Messages sent to the main area are also sent to stations that were specified in the subarea according to the type of the subarea. If a station has been specified in both the main area and a subarea or in two subareas to a main area, there is the possibility of a message being sent twice to that station. GEMCOS/UTILITY will warn of this possibility but will not mark it an error.

#### Example:

```

AREA ALLVEHICLES(1):
    STALIST = VEH0, VEH/NORTH,VEH/=.
AREA ROTARY TO1LAB2 (2):
    STALIST = ABCDEF, LOPIU, ABC123.
AREA TOALLABS (3):
    STALIST = STA1, STA2, TO1LAB2,

```

## OUTPUTROUTING SECTION

#### Syntax:

```

<output routing section> ::=
    OUTPUTROUTING: <routing list> /
    <empty>

<routing list> ::=
    <routing description> /
    <routing description> <routing list>

```

```
<routing description> ::=  
    <msgid routing statement>  
    <station list statement>
```

```
<msgid routing statement> ::=  
    MSGIDROUTING <msgid name> = <area name>.
```

```
<station list statement> ::= STALIST = <station name list>.
```

Semantics:

The <output routing section> must follow the STATION, AREA, and DEVICE sections. The <msgid name>, <area name>, and <station name>s must be previously specified in the TCL syntax.

The <output routing section> allows a message being sent to the originating station to be rerouted to an area of stations based on a message-ID. The MCS detects that the message-ID being sent to the originating station uses output routing and reroutes the message to the appropriate area.

Example:

```
OUTPUTROUTING:  
    MSGIDROUTING MSGID1 = AREA1.  
    STALIST = STA1,STA2.  
    MSGIDROUTING MSGID2 = AREA2.  
    STALIST = STA2.
```

In the above example, if a TP sends a message to STA2 with MSGID1 in COMMON [9], the message is sent to all stations in AREA1. If the message is sent to STA2 with MSGID2 in COMMON [9], the message is sent to all stations in AREA2. If the message is sent to STA1 with MSGID2 in COMMON [9], the message is sent only to STA1, since no output routing is specified for this combination of station and output message-ID.

## MESSAGE SECTION

Syntax:

```
<message section> ::= <message list> / <empty>
```

```
<message list> ::=  
    <message define> /  
    <message list> <message define>
```

```
<message define> ::=  
    MESSAGE <message name> = <message text>.
```

```

<message name> ::=
    INVALIDMESSAGEKEY / SECURITYERROR /
    UPDATESDISABLED / TRANSMISSIONERROR /
    BUSYMESSAGE / LOGONOK / LOGOFFOK /
    <mke>

<message text> ::=
    <message string> / <message text>, <message string>

<message string> ::=
    <EBCDIC code> <string> /
    <hexadecimal code> <hex string>

```

#### Semantics:

The <message section specifies replacement messages for standard system messages and/or one message that is accessible to all stations by a message key.

<Message name> INVALIDMESSAGEKEY corresponds to the message returned to a station when an input message does not contain a valid message key.

<Message name> SECURITYERROR corresponds to the message returned to a station when an input message contains a valid message key, but the user has violated access control restrictions.

<Message name> TRANSMISSIONERROR corresponds to the message returned to a station when an input message was not received properly.

<Message name> UPDATESDISABLED corresponds to the message returned when an input message for a MODIFY = TRUE program is entered while UPDATES have been disabled by a Network Control command.

<Message name> BUSYMESSAGE corresponds to the message returned to a <transaction mode> station when an input is attempted before a reply is received to the previous input.

<Message name>s LOGONOK and LOGOFFOK correspond to the messages sent to a station following successful logon or logoff.

<Message name> <mke> specifies a "canned" message that is recallable by <mke> from a station.

#### Example:

```

MESSAGE INVALIDMESSAGEKEY = "INVALID MESSAGE KEY".
MESSAGE SECURITYERROR = "INVALID MESSAGE KEY".
MESSAGE TRANSMISSIONERROR = "TRANSMISSION ERROR PLEASE RE-SEND".
MESSAGE 77 = "TELETYPE TEST MESSAGE", 4"OD25",
            "ABCDEFGHJKLMNOPQRSTUVWXYZ - 123456789".

```

## FILE GENERATION

GEMCOS/UTILITY uses and/or generates certain files to accomplish its tasks. These files may be label-equated to suit user needs. The following is a partial list of the internal names, external names, and file descriptions of the files needed to initiate a system. A list of all files is provided in appendix B.

Internal Name	External Name	File Description
CARD	CARD	Card reader, 80-character/ record source images.
LINE	LINE	Line printer, 120-character / record source and <list> output.
TAPE	DATAKOM/ DEFINITION/ DECK	Disk, 14-word records, blocked 30 library source images.
NEWTAPE	DATAKOM DEFINITION/ NEWDECK	Disk, 14-word records, blocked 30 <new> library source images.
CQUS	DATAKOM/ QUEUE/ CONTROL	Disk, 30-word records, unblocked Queue Control file.
IQUS	DATAKOM/ QUEUE/INPUT	Disk, unblocked Input Queue file. The default record size is 30 words.
OQUS	DATAKOM/ QUEUE/OUTPUT	Disk, unblocked Output Queue file. The default record size is 60 words.
FMTFILE	DATAKOM/ QUEUE/FORMATS	Disk, 30-word records, unblocked Format file.
IQUSDUP	DATAKOM/ QUEUE/ DUPLICATE/ INPUT	Optional Duplicate Input Queue file. Attributes are same as IQUS.
OQUSDUP	DATAKOM/ QUEUE/ DUPLICATE/ OUTPUT	Optional Duplicate Output Queue file. Attributes are same as OQUS.
CQUSDUP	DATAKOM/ QUEUE/ DUPLICATE/ CONTROL	Optional Duplicate Control file. Attributes are same as CQUS.

The following are the control cards necessary to generate a set of tables describing the data communications environment. All generated files are label-equated. The \$ card options and patching capabilities available in the Utility program are similar to the ones used in Burroughs standard compilers and are defined in the syntax description which follows. Sequence numbers should appear in columns 73-80.

```
?RUN GEMCOS/UTILITY
?FILE CQUS(TITLE = MYFILE/CQUS)
?FILE IQUS(TITLE = MYFILE/IQUS)
?FILE OQUS(TITLE = MYFILE/OQUS)
?FILE FMTFILE(TITLE = MYFILE/FMT)
?DATA CARD
$SET LIST SINGLE
      (TCL DEFINITION DECK)
?END
```

The result of a TCL compilation is the generation of the files needed to drive the MCS. These are the Input Queue file, the Output Queue file, and the Control file. A Format file is generated if formats are specified.

Depending on user specifications, a hard-copy listing of the TCL definition deck plus a listing of the various tables and directories generate by the TCL compiler may be output.

## Option Control Cards

Syntax:

```
<option control card> ::=
    $ <option list> <option group list>

<option group list> ::=
    <option group> / <option group list> <option group> /
    <empty>

<option group> ::= <option action> <option list>

<option action> ::= SET / RESET / POP / <empty> /

<option list> ::=
    <option> / <option list> <option> /
    <empty>

<option> ::=
    LIST / NEW / FORMATS /SEQ <sequence base> <sequence
    increment> / MERGE / SINGLE / SEQERR /
    SYNTAX / PAGE / VOID /
    VOIDT / CHECK / PATCH <patch specification>
```

```

<patch specification> ::=
    NEWFORMATS / <generalized identifier>

<sequence base> ::= <integer> / <empty>

<sequence increment> ::= + <integer> / <empty>

```

#### Semantics:

The function of the <option control card>s in the MCS definition deck is to change the current state of various compilation options. Associated with each option is a 48-bit stack in which the history of the state of that option is stored. The bit at the top of the stack represents the current state of the option. As with any other push-down stack mechanism, stored information is retrieved on a last-in, first-out basis. An option whose default value is RESET is initially assigned a stack filled with zeros, whereas an option whose default value is SET is initially assigned a stack with a 1 on top and zeroes elsewhere. SET causes the option stacks corresponding to the options following it to be pushed down one bit and a 1 to be put on top of these stacks. RESET causes the option stacks corresponding to the options following it to be pushed down one bit and a 0 to be put on top of these stacks. POP simply causes the option stacks corresponding to the options following it to be pushed up one bit.

Control cards must have the \$ symbol appearing in either column 1 or in column 2. The options to be manipulated are specified following the dollar sign (\$), with one or more spaces following each option. No option may continue past column 72. \$ Control cards may be interspersed at any point within the source language inputs. \$ Control cards do not appear in a NEW source file unless the \$ symbol is in column 2.

An <option list> with no <option action> causes all options which appear on the list to be SET, and all options not on the list to be RESET. Exceptions to this are the \$ PAGE option, which only causes the page option to be set, and the \$ PATCH option which causes the inclusion of a patch file.

LIST is set prior to system definition, and all other options are RESET.

## TCL-Directing Options

CHECK. CHECK causes the source input to be sequence-checked with sequence errors listed.

FORMATS. FORMATS may be set before the start of the first function or format and POPped after the last format to specify to the utility where the functions and formats start and end while doing a NONFORMATS compile. See the description of the NOFORMATS <control task> for more information.

LIST. LIST causes a line printer output listing to be generated. If LIST is RESET, only syntax error messages and the offending cards are listed.

MERGE. MERGE causes primary input on CARD to be merged with secondary input on TAPE. If matching sequence numbers occur, the primary input overrides. If MERGE is RESET, secondary input is totally ignored.

NEW. NEW causes a new source file to be created on a serial disk file known as NEWTAPE. This file is coded in EBCDIC and is structured in 14-word records and 420-word blocks. Thus, it may later be used as input through the file TAPE.

PAGE. The appearance of the word PAGE causes the output listing to be skipped to the top of a new page.

PATCH. A \$ PATCH card in a compile of the main TCL source file initiates a presence check on the file title specified. If the file is present, it is included, and VOID is set until another \$ PATCH or \$ POP PATCH card is encountered. All files in the New Format Directory file, TCL/NEW/FORMATS, are included in the compile if the <patch specification> is NEWFORMATS.

Example:

```
FORMAT FOUT [UPDATE]
  (Page[1] : A10, T (FUNC1, A6, 2),X1,A6,
  PAGE[2] : A6, A6, A6,
  PAGE[3] : A6, A6, A6,).
$ PATCH FORMAT1
.
.
.
$ PATCH FORMAT2
.
.
.
$ POP PATCH
```

SEQ. When SEQ is SET, the line printer output listing and the new secondary source file contains new sequence numbers as defined by the <sequence base> and <sequence increment>. As each image is output, it is assigned a sequence number equal to the value of the <sequence base>. The <sequence base> is then increased by the <sequence increment>. If the <sequence base> and <sequence increment> are unspecified, a base of 0 and an increment of 100 are assumed.

SEQERR. SEQERR causes sequence errors to be flagged as errors if CHECK is set.



**SEQUENCE BASE AND INCREMENT.** Sequence Base contains the sequence number assigned to the source image that is output if SEQ is SET. As each image is output, the sequence base is increased by the sequence increment.

**SINGLE.** SINGLE causes the listing to be single-spaced. When SINGLE is RESET, the listing is double-spaced.

**SYNTAX.** SYNTAX is used to override the <control section> specification. Thus, a generate can be done, but the files are not written to disk.

**VOID.** If VOID is SET, all input from TAPE and CARD is ignored until VOID is RESET or popped into RESET state.

**VOIDT.** If VOIDT is SET, all input from TAPE is ignored until VOIDT is RESET or popped into RESET state.

## SECTION 4

# APPLICATION PROGRAMS

There are five classifications of application programs which can be defined in the Transaction Control Language (TCL). A program's classification defines the interface which it uses with GEMCOS and generally the types of tasks it performs. In addition to these five program classifications, there are two types of remote file interfaces; programs using these interfaces need not be declared in the TCL. The user may also have an optional procedure to alter transaction routing. A conversational capability is also provided.

### TRANSACTION PROCESSORS

The five program classifications include three types of Transaction Processors (TPs) as indicated by a <program classification> of USER, PROCESS, or PORT.

#### User Program Interface

A User program may be written in COBOL68 or ALGOL and must be compiled as a procedure with five parameters. These parameters are passed to the User program when the MCS initiates it. The first four parameters are events; the fifth is a computational array row. The parameters are:

1. **MASTEREVENT** – caused by the User program after it has placed an output message into the Common area for transmission to the network.
2. **SERVICEEVENT** – not a requirement of the interface. It is provided as a "hook" in the event the user wishes to employ a Service program to perform certain editing type functions not provided by the MCS, or to perform other functions, such as installation-written data base management, transparent to the User program. The event is caused by the User program when it requires the message in the Common area to be serviced.
3. **EDITOREVENT** – not a requirement of the interface. It is provided in case the user employs an Editor program. Editor programs are similar to Service programs, except that they receive input directly from the Message Control System (MCS) and, in turn, pass the input to the User programs. The event is caused by the User program when it passes control of a message back to the Editor program.
4. **USEREVENT** – caused by the MCS or an Editor after an input message is placed into the Common area to inform the User program that a message is to be processed. It is also caused by an Editor program or Service program to return control of the Common area to the User program after the Editor or Service program has been given control explicitly by the User Program.
5. **COMMONAREA** – a global MCS computational array row. The MCS associates one row of the array for each active User program. This row is used to pass control information and message text back and forth between the MCS and the associated User program and between the User program and its Editor and Service programs.

In addition, the MCS sets a User program's Task-value to 4 when it wants the program to go to End-of-Job. The User program must check its Task value each time the MCS wakes it up. If its Task-value is 4, a User program should not reference its COMMONAREA but proceed immediately through its EOJ logic.

Figures 4-1 and 4-2 show the basic declarations for COBOL and ALGOL User programs.

The COMMONAREA has a length as defined by the <common size statement> in the TCL program description. The first 23 words are defined by the MCS and are used: 1) by the MCS to pass information concerning the current input message and 2) by the User program to pass information concerning the current output message. The layout of the control portion of the COMMONAREA is described in appendix A.

DATA DIVISION

77 MASTEREVENT USAGE IS EVENT RECEIVED BY  
REFERENCE.

77 SERVICEEVENT USAGE IS EVENT RECEIVED BY  
REFERENCE.

77 EDITOREVENT USAGE IS EVENT RECEIVED BY  
REFERENCE.

77 USEREVENT USAGE IS EVENT RECEIVED BY  
REFERENCE.

01 COMMONAREA USAGE IS COMPUTATIONAL RECEIVED  
BY REFERENCE.

02 COMMON PICTURE 9(8) USAGE IS COMPUTATIONAL  
OCCURS 1000 TIMES.

PROCEDURE DIVISION

USING MASTEREVENT  
SERVICEEVENT  
EDITOREVENT  
USEREVENT  
COMMONAREA.

<program>

STOP RUN.

Figure 4-1. COBOL User Program Basic Declarations

```

PROCEDURE USER (MASTEREVENT,
                SERVICEEVENT,
                EDITOREVENT,
                USEREVENT,
                COMMONAREA);

EVENT  MASTEREVENT,
        SERVICEEVENT,
        EDITOREVENT,
        USEREVENT;

ARRAY  COMMONAREA[0];

BEGIN

        <program>

END.

```

Figure 4-2. ALGOL User Program Basic Declarations

The control information is filled by the MCS on input, and certain parts of the control information must be changed by a User program on output. Although there are many ways to control the output message, for most transactions the user needs to insert values only into WORD[1], WORD[7], and WORD[0]. Following is an outline of all the words which can be altered by the User program to tell the MCS what to do with the output message. A more detailed discussion of the various fields is contained in appendix A.

#### WORD[0]:

Access to the COMMONAREA is controlled by the various events described above and by information contained in Word[0] of the COMMONAREA itself. Word[0] is used to indicate who is or who should be in control of the COMMONAREA. The User program's control-bit value is 0. The Editor and Service programs' control-bit values are specified in the TCL. The MCS has a control-bit value of 1.

The last thing a User program should do before causing the Master-event is to set WORD[0] to 1. This gives the MCS immediate control of the Common row.

#### WORD[1]:

The contents of this word dictate to the MCS what to do with the current output message.

#### WORD[4]:

This word controls the destination and, to some degree, the form of the current output message. On input, WORD[4] is set up to return all responses to the originator. Any deviation from this must be caused by the User program.

#### WORD[6]:

One field of this word can be used to override the UPDATE characteristics of a format and allow it to be used as a DISPLAY-only format.

#### WORD[7]:

Various fields of this word contain the character length of the message text and the character offset (if any) from WORD[23]. The word also allows the output message to be sent without an audit.

#### WORD[8]:

This word contains the station bits of the originator of the message as defined in the TCL. The contents of this word may be changed by an Access Control program during a valid log-on or log-off attempt.

#### WORD[9]:

This word contains the output message-ID which indicates the format to be applied to the output message. The output message-ID is a maximum of six characters and must be left-justified with trailing spaces. If the contents of this word do not comprise a valid output message-ID, no output formatting is performed.

#### WORDS[11], [12], and [13]:

Normally this word contains the name of the station that originated the message. If the user wishes to route a message based on the name of the destination station, the station name is placed here. Using station-numbering conventions for output routing is more efficient than utilizing station names.

#### WORD[16]:

This word contains the WAITFORAUDIT bit which the User program may reset in order to release messages to a station prematurely. (Refer to CONVENTIONS FOR A WAITFORAUDIT = TRUE SPECIFICATION in Section 5.)

#### WORD[17]:

The User program must store the Data Base Sequence Number that it acquires in this word. (Refer to RESTART AREA in Section 5.)

#### WORD[18]:

This word contains the transaction-state bit which must be turned on whenever a User program enters transaction state. (Refer to table 5-1 and item f under CONVENTIONS FOR A RECOVERY = TRUE SPECIFICATION in Section 5.)

No other words of the COMMONAREA should be altered by a User program. A User program waiting for an input message should sleep on its User-event. When this event is caused by the MCS, the User program should wake up, reset its USEREVENT, check its TASKVALUE, and then look at the COMMONAREA to find out what to do. While processing the input message, the User program may temporarily pass control of its COMMONAREA to the MCS in order to send secondary messages (WORD[1] of the COMMONAREA would have the value 2) to stations or to ask for some service of the MCS. If this is done, the User program must then sleep on its USER-EVENT until the MCS causes the event to give the program control of the COMMONAREA again. In the last pass of control to the MCS for an input message, WORD[1] of the COMMONAREA would have the value 3; this informs the MCS that the User program is finished with the current message and is waiting for the next one. The program should then go back to sleep waiting on its USEREVENT for the next input message.

## Process Program Interface

A Process program interface has been established for the more sophisticated user. This interface is more efficient than the User program interface and features additional capabilities, but it precludes the use of Editor and Service programs.

Process programs obtain and send data just as it appears within the internal queues of the MCS. A Process program must be written in DCALGOL (or DMALGOL if it accesses a DMS II data base) and must be compiled as a procedure with five parameters. These parameters are passed to the Process program when the MCS initiates it. The first parameter is an array, the next two parameters are events, and the last two are procedures. The parameters are:

1. CONTROLWORDS – an array that controls a Process program's interface to SENDMESSAGE and GETMESSAGE. (This is described in detail in appendix C.)
2. MASTEREVENT – caused by the Process program to indicate that a message has been prepared for transmission to the network.
3. PROCESSEVENT – caused by the MCS to indicate that a message for the Process program has been received from the network.
4. GETMESSAGE – a procedure called upon by the Process program for service and to obtain the message from the Input queue.
5. SENDMESSAGE – a procedure called upon by the Process program for service and to place a message into the Output queue for transmission to the network.

In addition, the MCS sets a Process program's Task-value to 4 when it wants the program to go to End-of-job. The Process program must check its Task-value each time the MCS wakes it up. Before terminating, the Process program must set its Task-value to 1 and cause the MASTEREVENT. Figure 4-3 shows the basic declarations for a Process program.

```

PROCEDURE PROCESS (CONTROLWORDS,
                   MASTEREVENT,
                   PROCESSEVENT,
                   GETMESSAGE,
                   SENDMESSAGE);

ARRAY CONTROLWORDS(0);

EVENT MASTEREVENT,
      PROCESSEVENT;

PROCEDURE GETMESSAGE,
          SENDMESSAGE;

BEGIN

    <program>

END.

```

Figure 4-3. Basic Declarations for a Process Program

The Process program sleeps on its PROCESSEVENT waiting for input. When the MCS realizes that it has input for a Process program, it causes the PROCESSEVENT. The PROCESSEVENT wakes up the Process program which can then obtain the input in a DCALGOL queue by passing the queue as a parameter to GETMESSAGE. The Process program uses the DCALGOL REMOVE function to move the message from its queue into a local storage area.

When the Process program has output to send, it uses the SENDMESSAGE procedure. When the Process program has finished processing an input message and has sent the last output, it should set its Task-value to 3 and cause the MASTEREVENT to tell the MCS that it is ready for another input. The Process program then goes back to sleep waiting for the next input.

Process programs may send output via SENDMESSAGE after they have already indicated to GEMCOS that they are finished processing the current transaction. The transaction must be a secondary type, and the WAITFORAUDIT bit must be reset. The program should not set its Task-value to 3 or cause the MASTEREVENT after using the SENDMESSAGE procedure.

## GETMESSAGE Procedure

The GETMESSAGE procedure of the MCS is the vehicle used to obtain input messages for TPs. This procedure is passed as a parameter to Process programs and, therefore, an explanation of its functions is presented here. GETMESSAGE expects two parameters: an array of control information and a DCALGOL queue.

The array of control information must be filled by the program to tell GETMESSAGE exactly what to do. During execution, GETMESSAGE fills this control array with pertinent data about the task it has just performed.

GETMESSAGE places the desired input message in the DCALGOL queue and then returns control to the caller. The DCALGOL queue may have several pieces of data in it depending upon the number of records required to hold the input transaction.

Each message in the DCALGOL queue is a piece of an entire message. Each piece contains a header with information about the data. The actual text of the message follows the header and contains message keys and valid data. A description of the header information in each piece of the message appears in appendix B as the Input Queue file description layout.

The layout of the array of control information, which controls the action of GETMESSAGE, is provided in appendix C.

## SENDMESSAGE Procedure

The SENDMESSAGE procedure of the MCS is the vehicle used to send output messages to stations. This procedure is passed as a parameter to Process programs. SENDMESSAGE expects two parameters: an array of control information and a pointer to the output text. The array of control information (this should be the same array passed to GETMESSAGE) must be filled by the program to tell SENDMESSAGE what to do. SENDMESSAGE moves the output text from the pointer and writes it to the output audit file. The layout of the array of control information which controls the actions of SENDMESSAGE is provided in appendix C.

## Port Program Interface

A Port-type program interface enables Transaction Processors to interface to the Network using the "Ports" feature of B 6000/B 7000 Systems. The presence of a Port-type interface program in the TCL generation initially requires DCALGOL compilation of the GEMCOS MCS to include Port interface code by setting the PORTS \$ option.



Port program conventions are basically the same as normal user TP conventions in GEMCOS, except for the following differences:

1. Port programs are processed by the MCS with no parameters. The Port program should contain a port file declaration of TPPORT. A sample COBOL74 Port file declaration is in figure 4-4. The MCS will process each Port program with the task file cards attribute set to the correct value for that program.
2. Anytime a User program waits or its USEREVENT or WAITFORAUDITEVENT, the Port Program waits on a read of its Port file.
3. Whereas a User program is passed the Common row that contains the transaction message, the Port program reads its Port file. The Port file should have a 01 level corresponding to that in Figure 4-4.
4. When a User program is ready to give control back to the MCS (for primary output, secondary output, or other MCS requests), it sets up the Common row and causes the MASTEREVENT. When a Port program wants the same functions performed, it sets up the Port file area and writes to the Port. With both types of interface, the program is returned control either as a result of the request, or when the next message is available.
5. A recoverable Port TP must use WAITFORAUDIT. A WAITFORAUDIT User TP tests COMMON[16].[0:1] to see if it must wait on the WAITFORAUDIT after sending its primary output but before exiting transaction state. A WAITFORAUDIT Port program performs the same function by first writing the primary output to its port. It then checks the low order digit at WORD [16] and, if set, reads a message from the Port. This message indicates audit is complete and also contains an updated DBSN in WORD [17] to be stored in the restart data set. It then exits transaction state and returns to the main loop.
6. Port programs may not use a Service program.

Some of the fields in the Port file description are in different words and/or bits than the User common description. To provide a guide as to the purpose of each field, a mapping is provided below between the name given in the MSG-area description in figure 4-4 and the field description in Appendix A.

MSG-AREA FIELD NAME:	SEE FIELD DESCRIPTION IN APPENDIX A FOR:
MSG-CONTROL	COMMON[0]
MSG-RECOVERY	COMMON[6].[42:2]
MSG-RETRY	COMMON[6].[45:3]
MSG-RERUN	COMMON[6].[46:1]
MSG-PRODISABLED	COMMON[6].[47:1]
MSG-PASSER	COMMON[1].[26:3]
MSG-DISPLAY	COMMON[6].[40:1]
MSG-ACTION	COMMON[1].[23:24]
MSG-STA-NO	COMMON[2].[23:24]
MSG-MODE	COMMON[3].[47:1]
MSG-ASSIGN	COMMON[3].[44:1]
MSG-ORIOBJORSTA	COMMON[3].[43:1]
MSG-MSGTYPE	COMMON[3].[42:3]
MSG-RETRTEHDR	COMMON[4].[43:1]
MSG-OVRRTTEHDR	COMMON[4].[42:1]

<b>MSG-AREA FIELD NAME:</b>	<b>SEE FIELD DESCRIPTION IN APPENDIX A FOR:</b>
-----------------------------	---

MSG-ORINUM	COMMON[3].[17:18]
MSG-TITLE	COMMON[4].[41:2]
MSG-DESTTYPE	COMMON[4].[39:8]
MSG-CRLF	COMMON[4].[31:8]
MSG-DEST	COMMON[4].[19:20]
MSG-DATE	COMMON[5].[47:24]
MSG-TIME	COMMON[5].[23:24]
MSG-EDIBIT	COMMON[6].[35:8]
MSG-PROG-NUM	COMMON[6].[7:8]
MSG-FMTERR	COMMON[7].[47:1]
MSG-LENGTHIN	COMMON[7].[46:47]
(ALSO RESULT INDICATORS	COMMON[7]
REQUEST RESULTS	COMMON[7] )
MSG-QBYPASS	COMMON[7].[46:1]
MSG-OFFSET	COMMON[7].[39:20]
MSG-LENGTHOUT	COMMON[7].[19:20]
MSG-STABITS	COMMON[8]
MSG-MESSAGE-ID	COMMON[9]
MSG-RESTART-DATA	COMMON[10]
MSG-REL-IQU	COMMON[10].[7:8]
MSG-STA-NAMEWORD	COMMON[11] THRU COMMON[13]
MSG-MODULE	COMMON[14]
MSG-FUNCTION	COMMON[15]
MSG-USER-ID	COMMON[16].[47:8]
MSG-ITEM-COUNT	COMMON[16].[39:16]
MSG-DUPAUDIT	COMMON[16].[1:1]
MSG-WAITAUDIT	COMMON[16].[0:1]
MSG-DBSN	COMMON[17]
MSG-TRANSTATE	COMMON[18].[0:1]
MSG-MTCHAR	COMMON[19].[47:8]
MSG-MTDEL	COMMON[19].[39:1]
MSG-PRIORITY	COMMON[6].[39:1]
MSG-SSN	COMMON[3].[39:22]
MSG-CONV-SIZE	COMMON[20].[47:12]
MSG-CONV-END	COMMON[4].[23:1]
MSG-CONV-DEALLOC	COMMON[4].[22:1]
MSG-CONV-CORESTORE	COMMON[4].[21:1]
MSG-CONV-DISKSTORE	COMMON[4].[20:1]
MSG-USERID1	COMMON[21]
MSG-USERID2	COMMON[22]

```

FD TPORT.
* THIS TP IS DECLARED IN THE TCL WITH COMMONSIZE = 343
01 TPCOMMON-AREA PIC X(343).
01 COMMON-AREA COMP.
03 COMMON-CONTROL.
05 WORDSOTHRU22 OCCURS 23 TIMES PIC 9(12).
03 COMMON-TEXT.
05 COMMONCHAR OCCURS 320 TIMES PIC 9(12).
*****
* OUTPUT MESSAGE AREA *
*****
01 MSG-AREA.
03 MSG-CONTROL-AREA.
05 MSG-CONTROL PIC 9(12) COMP.
05 MSG-WORD1 PIC 9(12) COMP.
05 MSG-ACTIONWRD REDEFINES MSG-WORD1.
07 MSG-RECOVERY PIC 9 COMP.
07 MSG-RETRY PIC 9 COMP.
07 MSG-RERUN PIC 9 COMP.
07 MSG-PRODDISABLED PIC 9 COMP.
07 MSG-PASSER PIC 9 COMP.
07 MSG-DISPLAY PIC 9 COMP.
07 MSG-ACTION PIC 9(6) COMP.
05 MSG-WORD2 PIC 9(12) COMP.
05 MSG-ORIWORD REDEFINES MSG-WORD2.
07 FILLER PIC 9(4) COMP.
07 MSG-STA-NO PIC 9(8) COMP.
05 MSG-WORD3 PIC 9(12) COMP.
05 MSG-VARIABLEWRD REDEFINES MSG-WORD3.
07 MSG-MODE PIC 9 COMP.
07 MSG-ASSIGN PIC 9 COMP.
07 MSG-OPIDBJORSTA PIC 9 COMP.
07 MSG-MSGTYPE PIC 9 COMP.
07 MSG-RETRTEHDR PIC 9 COMP.
07 MSG-OVRTEHDR PIC 9 COMP.
07 MSG-ORINUM PIC 9(6) COMP.
05 MSG-WORD4 PIC 9(12) COMP.
05 MSG-DESTWRD REDEFINES MSG-WORD4.
07 MSG-TITLE PIC 9 COMP.
07 MSG-DESTTYPE PIC 99 COMP.
07 MSG-CRLF PIC 9 COMP.
07 FILLER PIC 99 COMP.
07 MSG-DEST PIC 9(6) COMP.
05 MSG-WORD5 PIC 9(12) COMP.
05 MSG-DATETIMEWRD REDEFINES MSG-WORD5.
07 MSG-DATE PIC 9(6) COMP.
07 MSG-TIME PIC 9(6) COMP.
05 MSG-WORD6 PIC 9(12) COMP.
05 MSG-PROGWORD REDEFINES MSG-WORD6.
07 FILLER PIC 9(6) COMP.
07 MSG-EDIBIT PIC 9(3) COMP.
07 MSG-PROG-NUM PIC 9(3) COMP.

```

Figure 4-4. MSG-Area Description (Sheet 1 of 2)

```

05 MSG-WORD7 PIC 9(12) COMP.
05 MSG-LENGTHINWORD REDEFINES MSG-WORD7.
07 MSG-FMTERR PIC 9 COMP.
07 MSG-LENGTHIN PIC 9(11) COMP.
05 MSG-LENGTHOUTWORD REDEFINES MSG-WORD7.
07 MSG-QBYPASS PIC 9 COMP.
07 MSG-OFFSET PIC 9(6) COMP.
07 MSG-LENGTHOUT PIC 9(5) COMP.
05 MSG-STABITS PIC X(6).
05 MSG-MESSAGE-ID PIC X(6).
05 MSG-RESTART-DATA PIC X(6).
05 MSG-RESTARTNUM REDEFINES MSG-RESTART-DATA.
07 FILLER PIC 9(10) COMP.
07 MSG-REL-IQU PIC 9(2) COMP.
05 MSG-WORD11THRU13 PIC X(18).
05 MSG-STA-NAMEWORD REDEFINES
MSG-WORD11THRU13.
07 MSG-STA-NAME-CHAR PIC X
OCCURS 18 TIMES.
05 MSG-MODULE PIC 9(12) COMP.
05 MSG-FUNCTION PIC 9(12) COMP.
05 MSG-WORD16 PIC X(6).
05 MSG-AUDITWRD REDEFINES MSG-WORD16.
07 MSG-USER-ID PIC X.
07 MSG-ITEM-COUNT PIC 9(5) COMP.
07 FILLER PIC 9(3) COMP.
07 MSG-DUPAUDIT PIC 9 COMP.
07 MSG-WAITAUDIT PIC 9 COMP.
05 MSG-DBSN PIC X(6).
05 MSG-TRANSSTATE PIC 9(12) COMP.
05 MSG-WORD19 PIC 9(12) COMP.
05 MSG-SSNWORD REDEFINES MSG-WORD19.
07 MSG-MTCHAR PIC X.
07 MSG-MTDEL PIC 9 COMP.
07 MSG-PRIORITY PIC 9 COMP.
07 MSG-SSN PIC X(4).
05 MSG-WORD20 PIC 9(12) COMP.
05 MSG-CONVWORD REDEFINES MSG-WORD20.
07 MSG-CONV-SIZE PIC 9(4) COMP.
07 MSG-CONV-END PIC 9 COMP.
07 MSG-CONV-DEALLOC PIC 9 COMP.
07 MSG-CONV-CORESTORE PIC 9 COMP.
07 MSG-CONV-DISKSTORE PIC 9 COMP.
07 FILLER PIC 9(4) COMP.
05 MSG-USERID1 PIC X(6).
05 MSG-USERID2 PIC X(6).
03 MSG-TEXT-AREA COMP
05 MSG-TEXT OCCURS 320 TIMES PIC 9(12).
03 MSG-TEXTCHAR REDEFINES MSG-TEXT-AREA.
05 MSG-TEXT-CHAR PIC X
OCCURS 1920 TIMES
INDEXED BY MSG-TEXT-INDEX1
MSG-TEXT-INDEX.

```

## Editor Program Interface

The fourth program classification is invoked in the TCL by a <program classification> of Editor. The Editor program interface was established for users who wish to manipulate the input message prior to transmission of the message to a User program. An Editor program must be written in DCALGOL (or DMALGOL if it accesses a DMS II data base) as a procedure with eight parameters.

These parameters are passed to the Editor when the MCS initiates it. The first three parameters are events; the fourth is an event array; the fifth is a 2-dimensional computational array; the sixth, a queue array; the seventh, a procedure; and the eighth, a single-dimension array. The parameters are:

1. **MASTEREVENT** – caused by the Editor program if it places an output message into the COMMONAREA for transmission back to the network.
2. **EDITOREVENT** – caused by the MCS after an input message is placed into an Editor queue to inform the Editor program that a message is to be processed.
3. **USERTOMEVENT** – an event passed to User programs which they cause in order to pass control back to the Editor. (A port program passes control back to the Editor by placing the Editor's control-bit value in COMMON[0] and writing to its port. The MCS then gives control of the COMMON row to the Editor by causing the EDITOREVENT).
4. **USEREVENTARRAY** – an array of the USEREVENTS of all User programs in the TCL. The appropriate event within the array is caused by the Editor when it has finished manipulating the data in the COMMONAREA and wishes to pass control to the User program. The index into the array is contained in WORD[6].[19:8] of the COMMONAREA for User TP's. This array is not used for port TP's. The EDITOR passes control to the port TP by setting WORD[1] of COMMON to its regular action value plus 20 (i.e., 25 instead of 5, 36 instead of 16, 39 instead of 19), WORD[0] to 1, and causing the MASTEREVENT. This causes the MCS to write the message to the program's port file with the value 5, 16 or 19 in WORD[1] of COMMON.
5. **COMMONAREAS** – a 2-dimensional array of the COMMONAREAS of all User programs defined in the TCL. An Editor must scan all rows of the array looking for rows which belong to associated User programs and which contain the Editor's control-bit value in WORD[0].
6. **DATAQUEUES** – an array of queues, one queue for each User program, through which the Editor receives input messages. The index into this array is the array row number of the current row of the COMMONAREAS array on which the Editor is currently working.
7. **FORMATTER** – the input formatting module of the MCS. This is passed to all Editors so that they can format the input message prior to passing it to the User program.
8. **EDTDSC** – a single-dimension array which duplicates the data in all COMMON-WORD[0]'s and contains Port and size information. The Editor scans this array for its control-bit value prior to its scan of COMMON.

Additional information is passed to an Editor in its Task-value. Field [36:8] contains the Editor's control-bit value as specified in the TCL. This value is the control-bit value the Editor must look for in the EDTDSC and then in MSG-CONTROL (WORD[0]) of the COMMONAREA. The MCS sets the COMMONAREAs the Editor must scan. The MCS sets the Editor's Task-value to 4 when it wants the Editor to go to End-of-Job. The Editor must check its Task-value each time it resets its EDITOREVENT.

Figure 4-5 shows the basic declaration for an Editor program.

An Editor program can handle more than one program. If a program elects to have an Editor program, the MCS gives all input messages for the program directly to the Editor program. The Editor program must then pass the input on to the program. Since the Editor program can work on several programs, it must have access to all common rows and User-events and must have an EDITOREVENT which wakes it up. The Editor program also has an array of DCALGOL queues in which to receive inputs from the MCS and the MASTEREVENT to wake up Process-everything.

```
PROCEDURE EDITOR (MASTEREVENT,  
                  EDITOREVENT,  
                  USERTOMEVENT,  
                  USEREVENTARRAY,  
                  COMMONAREAS,  
                  DATAQUEUES,  
                  FORMATTER,  
                  EDTDSC);  
  
  EVENT MASTEREVENT,  
        EDITOREVENT,  
        USERTOMEVENT;  
  EVENT ARRAY USEREVENTARRAY [0];  
  ARRAY COMMONAREAS [0,0];  
  QUEUE ARRAY DATAQUEUES [0];  
  BOOLEAN PROCEDURE FORMATTER;  
  ARRAY EDTDSC [0];  
  
  BEGIN  
    <program>  
  END.
```

Figure 4-5. Basic Declaration for an Editor Program

The Editor program receives input from the MCS in a DCALGOL queue, processes it and places it in the program's Common row. The processing which the Editor does on the input message can involve using the MCS procedure FORMATTER which is passed to the Editor as a parameter. Once the Editor is finished with an input message, it causes the correct User-event, and the User program takes over. For a Port program, the Editor program passes the message back to the MCS which then writes it to the correct Port file, and the Port program takes over.

Since a User program has the EDITOREVENT, it too can wake up its Editor at any time to do additional work on a message. The Editor wakes up but does not know which User program needs assistance. It finds this out by checking EDTDSC for a specific control-bit value. This control bit value is defined for an Editor in the TCL. It must be placed in WORD[0] by a User program before causing the EDITOREVENT. Therefore, the Editor first checks EDTDSC for its value and, if found, it checks the associated COMMONWORD[0] for its value. Once the Editor finds the correct Common row, it performs its duty and causes the User-event (or in the case of a Port program, sends it back to the MCS which writes it back to the Port program). The Editor removes its control-bit value from WORD[0] and replaces it with the user's control-bit value, which is 0. This interplay between program and Editor can continue until the message is properly processed. When the message is ready to be sent to the MCS, WORD[0] must be set to 1 (the MCS's control-bit value), and the MASTEREVENT must be caused (or the Port file written) to wake up Process-everything. This action may be performed by the Editor or the program.

When an Editor wakes up and finds a row of the COMMONAREAS to be processed, it must determine what to do with the row. By convention, the MCS places the value 7 into WORD[1] of the COMMONAREA row and into EDTDSC[I].[45:1] (where I is the common row index) to indicate that this is a new input message and that the Editor must get the message from the appropriate queue. The user may devise passes of control conventions between the program and Editor as long as they do not conflict with the MCS's conventions.

A skeletal Editor program is supplied on the release tape to provide users with the standard input-formatting capability. In its supplied form, it can process any input message by calling upon FORMATTER. However, any other processing of messages, or any pass of control to the Editor from a User program, must be added to the supplied Editor program.

## Service Program Interface

The fifth program classification is invoked in the TCL by a <program classification> of Service. The Service program interface was established to allow the user to perform certain editing type functions not provided by the MCS. It can also perform functions such as data base management, transparent to the User program. Port programs may not make use of a Service program.

Like an Editor, a Service program can handle many User programs. Unlike the Editor, however, it never receives a message from the MCS; it does not have any DCALGOL queues or the procedure FORMATTER passed to it. A Service program may be written in COBOL or ALGOL and must be compiled as a procedure with four parameters. These parameters are passed to the Service program when the MCS initiates it. The first two parameters are events, the third is an event array, and the fourth is a 2-dimensional computational array. These parameters are:

1. MASTEREVENT – can be caused by the Service program if it places an output message into the COMMONAREA for transmission back to the network.
2. SERVICEEVENT – caused by the User program when it desires a message in the COMMONAREA to be serviced by the Service program.
3. USEREVENTARRAY – an array of the USEREVENTS of all User Programs defined in the TCL. The appropriate event within the array is caused by the Service program when it has finished manipulating the data in the COMMONAREA and wishes to pass control back to the User program. The index into the array is contained in WORD[6].[19:8] of the COMMONAREA. (This index is base 0 and must be incremented by 1 if the Service program is written in COBOL.)
4. COMMONAREAS – a 2-dimensional computational array of the COMMONAREAS of all User programs defined in the TCL. A Service program must scan all rows of the array looking for rows which contain the Service program's control-bit in WORD[0].

Additional information is passed to a Service program in its Task-value. Field [36:8] contains the Service program's control-bit value as specified in the TCL. This value is the control-bit value that the Service program must look for in WORD[0] of each row of the COMMONAREAS array. Field [7:8] contains the number of rows of COMMONAREAs (base 1) the Service program must scan for its control-bit value. The MCS sets the Service program's Task-value to 4 when it wants the Service program to go to End-of-Job. The Service program must check its Task-value each time it resets its SERVICEEVENT.

Figures 4-6 and 4-7 show the basic declarations for COBOL and ALGOL Service programs.

A Service program sleeps on its SERVICEEVENT until a User program causes its event. The Service program wakes up, resets its event, and starts to scan all of the rows of COMMONAREAS searching for its control-bit value in WORD[0]. For each such row it finds, the Service program services the message and, generally, passes control of the row back to the User program by moving 0 to WORD[0] of the COMMONAREA and causing the appropriate event within the USEREVENTARRAY. The Service program may also send messages to areas or stations or ask for other MCS services. The COMMONAREA layout in appendix A provides the appropriate control fields.

A Service program is not supplied with the MCS. Therefore, all processing and passes of control to the Service program must be written by the customer. The source code of a sample Service program is included on the release tape.

#### NOTE

The Service program interface is not available for Port type programs.



DATA DIVISION

77 MASTEREVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 SERVICEEVENT USAGE IS EVENT RECEIVED BY REFERENCE.

01 USEREVENTARRAY USAGE IS EVENT  
RECEIVED BY REFERENCE.  
02 EVENTARRAY OCCURS 128 TIMES.

01 COMMONAREAS USAGE IS COMPUTATIONAL  
RECEIVED BY REFERENCE.  
02 COMMONARRAY OCCURS 128 TIMES.  
03 COMMON PICTURE 9(8) OCCURS 1000 TIMES.

PROCEDURE DIVISION.

    USING MASTEREVENT  
        SERVICEEVENT  
        USEREVENTARRAY  
        COMMONAREAS.

    <program>

STOP RUN.

Figure 4-6. Basic COBOL Declarations for a Service Program

PROCEDURE SERVICE (MASTEREVENT,  
SERVICEEVENT,  
USEREVENTARRAY,  
COMMONAREAS);

EVENT MASTEREVENT,  
SERVICEEVENT;  
EVENT ARRAY USEREVENTARRAY [0];  
ARRAY COMMONAREAS [0,0];

BEGIN

END.

Figure 4-7. Basic ALGOL Declarations for a Service Program

## OBJECT JOB INTERFACE

A remote file interface has been established within GEMCOS in the form of the Object job application program. An Object job is an externally compiled program which can communicate with a particular station as well as with any and all TPs defined in the TCL.

Object jobs communicate with the rest of the network by reading and writing into a file whose KIND attribute has been declared to be REMOTE and which has been file-equated to a valid station within the network. An Object job may be attached to multiple stations by declaring each one as a separate remote file; however, each station in the network may be attached to only one Object job at a time. Object jobs are not declared in the TCL. If Object jobs are to be used, an Object I/O key must be defined in the GLOBAL section (refer to section 3). This Object I/O key is a 1-character key which must precede each message between an Object job and the station to which it is attached.

Messages transmitted from the station or sent out by the Object job without the Object I/O Key are routed to the appropriate TP based on the message key. Responses from the TP are routed back to the originator (station or Object).

All messages sent by the Object job, either to its associated station or to a TP, must be terminated by a 4"FF" in the last character. All responses from a TP back to an Object job are received in pieces of 80 characters each, or up to the next Carriage Return character (4"OD") in the message. Each piece of the response is obtained by reading the remote file. The last piece is marked by 4"FF" in the last character. If the last piece is exactly 80 characters, the 4"FF" will be in the eighty-first character.

## BATCH JOB INTERFACE

Batch jobs are a specific subset of Object jobs which can route messages only to TPs. Batch jobs must follow the conventions outlined below; in return, they are addressed by the GEMCOS synchronized recovery mechanism.

A station named BATCHLSN must be declared in NDL. All Batch jobs must communicate with the MCS by reading and writing a remote file containing only this station.

Prior to routing any messages to a TP, the batch job must perform a handshake with the MCS by writing a record with the following format:

1. The first character of the handshake message must be a reserved message key, defined as the Batch I/O key in the TCL <global section> which identifies the message as a Batch job handshake.
2. The next three characters of the handshake message should contain the system number in decimal form. The system number is defined in the TCL and facilitates the MCS's routing of Batch-job-to-TP messages.

3. After writing the handshake message to the MCS, the Batch job should read its remote file in order to receive the MCS's response to the handshake. One of the following responses is then returned:
  - a. NO in the first two characters of the response to indicate the handshake was denied. This occurs if the number of Batch jobs exceeds the maximum as specified in the GLOBAL section of the TCL.
  - b. Computational zero in the first word of the response to indicate that the Batch job should execute normally without recovery.
  - c. Computational nonzero in the first word of the response to indicate that the Batch job should execute its recovery code (refer to section 5).

After a successful handshake, the Batch job may begin sending messages to TPs. The first four characters of each transaction initiated by a Batch job comprise a numeric field containing a user-supplied transaction counter. Typically, this counter might be a pointer into a serial file being processed by the Batch job. This transaction counter is used in the recovery mechanism and is further explained in section 5.

As with Object jobs, all messages sent by the Batch job (including the handshake) must be terminated by 4"FF", and all responses are received in a maximum of 80-character pieces. The last piece is terminated by 4"FF". If the last piece is exactly 80 characters, then 4"FF" will be in the eighty-first character.

The actual message which is to be routed to a TP (i.e., the message key and associated text) should appear immediately following the first four characters of the message. These first four characters are stripped from the message before it is presented to the TP.

Before sending an input transaction to the MCS (i.e., writing to its remote file), a Batch job must first receive a response (i.e., read of its remote file) to the previous input transaction submitted. This one-in, one-out convention is used to regulate the rate at which Batch jobs may submit input transactions to the MCS. This regulation becomes a function of the message volume from the "live" network. It is these messages with which the Batch jobs transactions are competing. The goal here is to prevent a Batch job from flooding a TP Input queue with transactions at the expense of response time to the "live" network. In addition, a one-in, one-out message flow is necessary for complete recovery integrity.

Any number of Batch programs up to the maximum declared in the TCL may be running at the same time. Every Batch program must be reading and writing the same remote file. This is the file which contains the station BATCHLSN.

## ALTERNATIVE TO THE BATCH INTERFACE

The batch interface feature of GEMCOS enables the user to process batch transactions concurrently with real-time transactions and still have the synchronized recovery capability. Since the fast response of the real-time transaction is of prime concern to the user, the Batch interface has been implemented in a manner to prevent it from inhibiting the real-time system. Some users have a need, during periods of low real-time transaction activity, to give the batch processing requirements equal priority. The alternative approach to batch processing allows the user to establish equal priority for the batch processing and maintain the synchronized recovery feature requirements.

The batch processing program is written following the normal conventions for a synchronized recovery transaction processor. The program is initiated by a station input transaction. The initiating station should not be switched to another system while the batch processing is in progress.

The initiating station input transaction should cause the batch processing program to do the following:

1. Establish a beginning transaction sequence number for recovery purposes in its own local storage area.
2. Locate the starting point in the batch input file.
3. Open the data base.
4. Get a GEMCOS DBSN.
5. Create and send a secondary message to the initiating station. This message will acknowledge the start transaction and free the station for other work.
6. Create and send a primary message TP-TO-TP NO WAIT to itself using the DBSN acquired. The text should contain the established transaction sequence number plus 1 and the batch input file location.

The normal transaction processing flow for the program is:

1. Receive GEMCOS input transaction.
2. Increment internal transaction sequence number.
3. Locate batch record based on information in the input transactions.
4. Perform data base update following the normal synchronized recovery conventions.
5. Create and send a primary message TP-TO-TP NO WAIT to itself, using the DBSN acquired. The text should contain the current transaction sequence number plus 1 and the batch input file record location.

The recovery initiation of a batch processing program can be determined by checking the status of the recovery indicator in the Common area. When this type of initiation occurs, the program should:

1. Store the input message transaction sequence number minus 1 in the program's local transaction sequence number.
2. Enter the normal transaction processing flow of the program.

The recovery transaction input message should be handled as follows:

1. If the input message transaction sequence number of the transaction is greater than the program's local transaction sequence number, then enter the normal transaction flow.
2. If the input transaction sequence number of the message is less than or equal to the internal transaction sequence number, the transaction should be ignored. This is done by sending a primary message to the originating station with a length of zero and a DBSN of zero.

The Common retry indicator must be observed during this type of processing. The rejection of an abort-causing batch transaction should follow the normal transaction flow, omitting step 4.

The batch processing program still processes real-time station input transactions during the TP-to-TP mode. The real-time input can be used to stop the batch processing or alter its normal flow (e.g. monitor transactions).

## USER ROUTING PROCEDURE

When using the station user control statement or the message-key user control statement, the INPUTUSERHOOK procedure is invoked. This procedure may be written by the user in DCALGOL. It should be compiled at level 4 and the object code bound into the MCS. Figure 4-8 is a sample of the procedure, to which the user may add the appropriate DCALGOL code in order to generate the desired transaction codes.

The sample routing procedure scans the input message text and generates transaction code, which is simply the original message key. It should insert this valid transaction code, followed by a period, into the array to which TRANCODE points. This transaction code is used by the MCS to route the message to the correct program. The maximum length of the transaction code is 23 characters.

If the user-written procedure is not bound into the MCS, and some form of user control is specified in the TCL, the default INPUTUSERHOOK procedure is processed.

```

PROCEDURE INPJUSERHOOK ( %
MSGTEXT,      % POINTER TO THE START OF THE MESSAGE
              % TEXT
MSGLENGTH,    % CHARACTER LENGTH OF THE MESSAGE TEXT
TRANCODE,     % POINTER TO TRANSACTION CODE PASSED
              % BACK TO GEMCOS AND USED TO DETERMINE
              % MESSAGE ROUTING AND FORMATTING. THE
              % TRANSACTION CODE IS A MAXIMUM 23
              % CHARACTERS LONG AND
              % IS TERMINATED BY A PERIOD
INPUTQUEUE,   % THE INPUT QUEUE TABLE USED BY THE
              % MCS. THE VALUES IN THIS TABLE MAY
              % BE USED BY THE USER TO DETERMINE
              % MESSAGE ROUTING
STACONV,      % THE ORIGINATING STATION'S
              % CONVERSATION AREA
MKES);        % THE DICTIONARY OF MESSAGE KEYS FOR
              % THIS SYSTEM

VALUE MSGTEXT,%
      TRANCODE,%
      MSGLENGTH;%
POINTER MSGTEXT,%
      TRANCODE;%
INTEGER MSGLENGTH;%
ARRAY INPUTQUEUE(0,0);%
ARRAY STACONV(0);%
ARRAY MKES(0);
BEGIN%
  POINTER PTEXT;%
  INTEGER LENGTH;%
  SCAN PTEXT:MSGTEXT FOR LENGTH:MSGLENGTH%
  WHILE EQL " "; %
  IF PTEXT GEQ "1" THEN%
    REPLACE TRANCODE BY PTEXT FOR
      MIN(23,MSGLENGTH)%
    WHILE GEQ "1","." %
  ELSE %
    IF PTEXT LSS "A" THEN %
      REPLACE TRANCODE BY PTEXT FOR 1,"." %
    ELSE%
      REPLACE TRANCODE BY PTEXT FOR
        MIN(23,LENGTH)%
      WHILE GEQ "A","." ;%
  END;%

```

Figure 4-8. Sample of User Routing Procedure

## CONVERSATIONAL INTERFACE

The GEMCOS Conversational capability allows a station to enter into a dialogue with a program. There are two TCL attributes related to this capability:

1. **CONVERSATIONAL** – This is a Boolean station attribute directing the station to participate in a dialogue with a program. When it is set to TRUE, a Conversational row is assigned to the station in anticipation of in-memory storage of a conversation. If conversations are never stored in memory, the row is never accessed. Deallocation of the row is under control of the user TP. The size of the row depends on the size of the Conversational area of the particular program in conversation with the station.
2. **CONVERSATIONSIZE** – This is a program attribute specifying the size of the program's Conversational area in words. The Conversational area is part of the Common area beginning in word 23 (the beginning of the text area for nonconversational programs) and extending for <conversation size> words. The Conversation area is null on the first message of the day and on the first message after the end of a conversation. Text immediately follows the Conversation area, and word 7 of the Common, the length word, does not include conversational data.

Storage of the conversation is under control of the user TP on primary output to the station. Bits in the Common row are examined on primary output to determine if the Conversation area should be stored in memory and/or on the audit trail, or if the conversation has ended and the Conversational storage area for the station should be deallocated. The following bits affect conversation storage and should be set on primary output.

1. **MSG-CONV-END (COMMON[4].[23:1])**: Value 1 indicates End-of-Conversation. The next input from the destination station produces nulls in the program's Conversational area.
2. **MSG-CONV-DEALLOC (COMMON[4].[22:1])**: Value 1 tells the MCS to deallocate the station's storage area. This word is examined only if MSG-CONV-END (COMMON[4].[23:1]) = 1.
3. **MSG-CONV-CORESTORE (COMMON[4].[21:1])**: Value 1 tells the MCS to store the program's Conversational area in the station's in-memory Conversational storage area.
4. **MSG-CONV-DISKSTOR (COMMON[4].[20:1])**: Value 1 tells the MCS to store the program's Conversational area on the station's audit trail.

When conversations are audited on the station's audit trail and the program participates in the GEMCOS synchronized recovery, the Conversational area for the program is also recovered and reflects the conversation as it existed for any reprocessed inputs.

When conversations are audited on the station's audit trail for checkpoint recovery programs, the Conversational area reflects the conversation as it existed after the last input that had a matching output in the audit trail.

## SECTION 5

# RECOVERY

B 5000/B 6000/B 7000 Series GEMCOS provides a broad range of recovery capabilities within the Transaction Control Language (TCL). This allows the user the flexibility to analyze application-oriented needs and then select the recovery options required.

Recovery is the re-establishment of normal processing. It is necessary when failure occurs, interrupting normal processing of transactions until some corrective action is taken.

It is assumed that the user wants to minimize exposure to situations where file and/or data base reloads are required following a failure. Under such conditions, the on-line user network normally remains in a nonproductive state while reprocessing takes place for all transactions that have occurred since the time the data base was last dumped to some form of backup storage. The time loss due to this nonproductive state is based directly upon several main factors:

1. Data base size.
2. Length of time since last data base dump.
3. Transaction volumes since last data base dump.
4. Transaction-throughput capacity.

Further, it is highly desirable to free user programmers from recovery concerns to the maximum extent possible. Recovery solutions are extremely complex for an integrated, batch/on-line system where a data base is being updated in a multiprogramming environment.

The goal of B 5000/B 6000/B 7000 Series GEMCOS is to make the recovery process transparent to application programs. In return, the user is expected to follow several straightforward programming conventions for normal processing. These conventions vary slightly, based upon the level of recovery required.

Although the Message Control System (MCS) was designed to be "ignorant" of any Data Management System, and contains no data management code embedded within it, the recovery mechanism fully accommodates DMS II (Burroughs Data Management System – Version 2). The recovery mechanism works with any Data Management System as long as the programming conventions discussed in this chapter are followed.

## NO RECOVERY

Among the recovery options available to the user is the specification that no recovery be applied to a "failed" system. This option is not a default option of any of the other TCL recovery options. The "no recovery" option must be explicitly specified for a system by means of the following TCL syntax in the <system section>.

```
FLUSHRECOVERY = TRUE
```

All Input and Output queues are flushed at the beginning of the recovery mechanism for that system. The result is that no unprocessed transactions are processed, and all undelivered output is discarded.



In addition, all programs in a system where `FLUSHRECOVERY = TRUE` are notified that a recovery mechanism was initiated. This notification message can be identified by a value of 7 in `MSG-RECOVERY (COMMON[3].[42:3])`. When programs which receive this restart message are ready to receive input from the network, they should give control of `COMMON` back to the MCS by generating a primary output. This primary output may be `NULL` (i.e., length = zero). Any output whose destination is the originator is directed to the network control stations.

## **CHECKPOINT RECOVERY**

In the present implementation, the MCS automatically generates an audit trail of input/output transaction images to either disk or disk pack. Generation of this audit trail is transparent to the application program. The audit trail is utilized by the MCS during the recovery process in order to re-establish its internal message queues. One relatively simple type of recovery, requiring no special programming conventions, is based on the MCS Checkpointing System. The MCS periodically writes information pertaining to the status of transaction flow to either disk or disk pack. These writes are called checkpoints. Checkpoints relate to the audit trail, and their frequency is controllable, to a large extent, in the TCL. Following a failure, the MCS re-establishes queue pointers based upon the last checkpoint.

Input queue pointers are positioned to the first input (after the last checkpoint position) which does not have a primary output message being audited. This is an attempt to eliminate as many duplicate messages at the stations as possible, and to eliminate unnecessary reprocessing of transactions. This level of recovery is sufficient for many applications (normally non-DMS installations).

## **SYNCHRONIZED RECOVERY**

A higher level of recovery is provided which calls for the MCS to synchronize application programs and transaction queues with the state of associated DMS II data bases following an automatic "rollback" resulting from a failure. The reason that DMS II rolls back a data base in time is to back out logical data base updates that were partially completed at failure time. (A single logical update transaction normally results in multiple data base updates.) During recovery the MCS performs an audit trail analysis relative to the rolled-back data base state, such that input transactions which were backed off the data base are "recycled" for application processing. Input transactions which were successfully audited by the MCS prior to failure, which have not yet reached the point where they are passed to an application program for processing, are requeued. Transactions falling into the "recycle" classification can be reprocessed so that the identical update sequence to the data base is maintained.

The MCS does not just hand to each program its particular transactions in the same sequence that the program received them prior to failure. Rather, if specified, the MCS attacks the more complex problem of reestablishing the exact sequence of events that occurred on the data base. This recovery technique considers the fact that some variable number of independent program executions could be receiving transactions in parallel, where each transaction could result in multiple updates to the data base in parallel. This feature can be extremely important when multiple transactions which result in access to common data can be asynchronously flowing through the system at the same time. Synchronized recovery is oriented toward allowing application programs and terminal operators to ride through failures without any need to know that such failures have occurred. At the same time, data bases can be maintained which are in total agreement with any messages delivered to the network prior to the time of failure. Additionally, for input transactions recycled because of a data base rollback, an analysis is performed by the MCS on application-generated output in order to eliminate duplicate transmissions to the network. The following messages are generated during a normal MCS synchronized recovery cycle:

Message	Meaning
*RECOVERY CYCLE INITIATED	Recovery begins.
*RESTART TP INVOKED	The user's restart TP has been invoked.
*RECOVERY STACK INVOKED	A separate stack for recovery has been invoked.
*RECOVERY STACK COMPLETED	The separate recovery stack goes to End-of-Job.
*RESTART TP COMPLETED	The restart TP was sent to End-of-Job.
*SYSTEM <number> RECOVERED	Recovery process ends.

## TCL Syntax

This type of synchronized recovery can be invoked for any Input queue by means of special Input queue syntax in the TCL. Two levels of synchronized recovery can be invoked depending upon the syntax used. Each level assumes that certain programming conventions are followed by the user.

The first level of synchronized recovery is invoked by the user by specifying RECOVERY = TRUE for an Input queue. This indicates that the programmer is going to follow several straightforward programming conventions. In return, the MCS attempts to synchronize the data-communications recovery with the data base recovery. In most instances, the MCS is successful; however, at this level of recovery, the MCS cannot guarantee that a fully synchronized recovery will take place. There can be input transactions recycled unnecessarily, possibly resulting in extraneous updates to a data base and/or there might be lost output messages. This may be perfectly acceptable in some environments (especially where duplicate data base updates are protected against by the designs of the application programs); in others, however, the results could be unfavorable.

To eliminate such potential risks, a second level of synchronized recovery can be invoked by additionally specifying WAITFORAUDIT = TRUE for an Input queue. If the user follows the programming conventions dictated by this syntax, the MCS provides a fully synchronized data base/data communications recovery.

## Recovery-Related Conventions

The conventions which must be followed by the user in order to ensure a synchronized recovery are detailed below.

### CONVENTIONS FOR A RECOVERY = TRUE SPECIFICATION

The following are the conventions for a RECOVERY = TRUE specification:

1. A User program must claim transaction-related resources prior to using any of them. In particular, this means locking all necessary data management records before entering transaction state. In many situations, this might entail locking just a particular node within a data hierarchy.
2. A User program must not unlock any transaction resources until the transaction is complete. In particular, this means allow the DMS II END-TRANSACTION to unlock data management records.
3. After transaction resources are locked, a User program should obtain a Data Base Sequence Number (DBSN) from the MCS and store it in the MSG-DBSN field of the COMMONAREA (Word 17). The program receives two additional parameters from the MCS: a LOCKEVENT and a pointer to a Global DBSN. To obtain its DBSN value, the program should lock the LOCKEVENT, increment the DBSN by one, move it to the COMMONAREA, and unlock the LOCKEVENT. The DBSN stamp normally goes in all output messages generated by the corresponding input transaction.
4. A User program must send messages to stations according to the following set of protocols:
  - a. "Secondary outputs" are sent first: These are outputs which are passed to the MCS with MSG-ACTION (WORD[1] of the COMMONAREA) set to 2. The value 2 in the action word signifies to the MCS that the User program is not finished with the current transaction and wishes to send more output. After extracting the appropriate information from the COMMONAREA, the MCS passes control back to the User program for further processing.
  - b. The last output generated during normal processing by a User program as a result of receiving a given input transaction is termed the "primary output." By strictest definition, the primary output of a transaction is a message sent by the User program with value 3 in MSG-ACTION and directed to the originating station. The value 3 in the action word signifies to the MCS that the User program is done with the current transaction. In actuality, the last output associated with an input transaction need not result in a message being delivered to the originating station. The last output (i.e., action word = 3) of a User program could have a length of 0 which results in no message being delivered, or the program could cause its last output to deliver a message to some station other than the originator. In both cases, the MCS automatically generates a "dummy" primary output for the originating station.
  - c. Generally, all outputs should be generated after the DBSN is obtained so that all output messages are stamped with it. There are instances when the user may want to generate an output before the DBSN is obtained; for instance, in order to shorten the time the program must be in DMS II transaction state. This may be done safely as long as all other guidelines for DBSN, primary output and WAITFORAUDIT (when WAITFORAUDIT = TRUE) are followed. User programs must set the MSG-DBSN field in the COMMONAREA to its maximum value (all bits = 1) in this case. This allows the message to participate in the output message analysis explained later in this section. When issuing secondary output after the DBSN is obtained, the "real" DBSN should be placed in the COMMONAREA.

5. A User program must cause the DMS II restart information to be forwarded to the DMS II audit trail when leaving transaction state, as opposed to when entering it.
6. A User program must inform the MCS whenever it passes into transaction state by setting the TRANSACTION STATE bit (Word [18].[0:1] to one for a USER program or TASK-VALUE to -1 for a Port program). Likewise, it must inform the MCS when it passes out of transaction state by resetting the TRANSACTION STATE bit to 0.
7. A User program must immediately pass control back to the MCS with MSG-ACTION (COMMONAREA WORD[1]) set to 20 whenever a DMS II abort exception is received. Remember, however, that the program must have control of its COMMONAREA row in order to accomplish the pass of control. In this case, the program should not wait on its WAITFORAUDIT event, even if the WAITFORAUDIT bit is one (value = 1).

#### CONVENTIONS FOR A RECOVERY = TRUE, WAITFORAUDIT = TRUE SPECIFICATION

The following are the conventions for a RECOVERY = TRUE, WAITFORAUDIT = TRUE specification:

1. If RECOVERY = TRUE and WAITFORAUDIT = TRUE syntax is used, an additional convention must be followed. This is a guarantee by the user that he or she will not exit from transaction state until the MCS had audited the primary output for the transaction. The MCS sets or resets the MSG-WAITAUDIT field (COMMONAREA WORD[16].[0:1]) for each transaction, based on whether the message key was defined in a WAITFORAUDIT = TRUE Input queue. If the bit is one, the User program should wait for the audit of the primary output of the current transaction before exiting from transaction state. The MCS passes an event to the User program to coordinate waiting for the audit.
2. In a WAITFORAUDIT = TRUE transaction, secondary messages are not released to their respective destination stations until the primary output is received by the MCS and is audited. Even though it ensures that messages received at a station reflect a completed transaction, this holding back of secondary messages might not be desirable in some instances. For instance, the User program might want to release the originating station from transaction mode before the primary output is sent.

The User program can cause any output message to be immediately released to its destination by setting the WAITFORAUDIT bit to zero. If this is output to the originating station, then that station is released from transaction mode. The use of this feature, however, does have some recovery implications. A message is released before the transaction is completed. This message, already received at the station could reflect some state of the data base which would be true if recovery were not necessary, but which is no longer true after recovery. This could happen if the original transaction had not obtained a DBSN before recovery was required. The order in which the transaction is reprocessed would be determined by the order in which it was originally entered into the system, not in the order of data base updating. If the user wishes any succeeding messages to follow the WAITFORAUDIT conventions, thus ensuring their integrity, the WAITFORAUDIT bit must be reset back to one.

The proper sequence of events for a User program which has RECOVERY = TRUE or both RECOVERY = TRUE and WAITFORAUDIT = TRUE specified for one of its Input queues is outlined in the following basic logic flow example.

```

INITIALIZE.
  OPEN DATABASE.
  INITIALIZE RESTART DATA SET.
SLEEP.
  WAIT AND RESET USEREVENT.
  LOCK DATABASE RECORDS TO BE UPDATED.
  SET TRANSACTIONSTATE BIT IN COMMON.
  BEGIN-TRANSACTION WITH NO-AUDIT.
    OBTAIN DBSN MOVE TO COMMON.
    MOVE RESTART INFORMATION TO RESTART AREA.
    .
    .
    .
  UPDATE DATABASE.
  SEND SECONDARY OUTPUTS.
  SAVE WAITFORAUDIT FLAG FROM COMMON.
  SEND PRIMARY OUTPUT.
  CHECK WAITFORAUDIT FLAG
    AND WAIT(AUDITEVENT) IF APPROPRIATE.
  END-TRANSACTION WITH AUDIT.
  RESET TRANSACTIONSTATE BIT IN COMMON.
  GO TO SLEEP.

```

Figure 5-1. Outline of the Basic Logic Flow of a RECOVERY=TRUE and/or WAITFORAUDIT=TRUE User Program

## Restart TP

Every system that has a RECOVERY = TRUE Input queue must have a Restart program defined. The Restart program is defined in the program section of the TCL, like other User programs, with the following special syntax:

```
RESTARTPROGRAM = TRUE.
```

Only one Restart program may be defined for a system.

When DMS II rolls back a data base, the Restart program is processed by the MCS as the first step in its synchronized recovery mechanism. The Restart program retrieves all of the data-base restart data of Transaction Processors (TPs) that are involved in the recovery mechanism, sorts it, and passes it to the MCS. The Restart program waits until notified by the MCS that recovery is complete, then deletes the restart data and goes to End-of-Job.

GEMCOS supplies a fully documented COBOL source file of a sample Restart program to be used with DMS II. This sample demonstrates the proper passes of control from the Restart program to the MCS, as well as the logic flow within the program. This sample source file can be patched by the user to replace the naming conventions of the restart data set used by GEMCOS with the user's data names.

## Restart Area

Each User program may have associated with it from one to three Input queues from which it receives input. Each program is required to save information about the transactions it processes out of each of its queues in the DMS II restart data set. Some information to be stored is supplied by the MCS in the control portion of the COMMONAREA passed to the User program. Other information is acquired as a result of the recovery conventions.

MSG-RESTART-DATA (WORD[10] of the COMMONAREA) contains data which must be stored in the restart data set. This word has four fields:

Field	Description
[47:8]	Input queue number as defined in the TCL in which this message resides.
[39:8]	An index into the MCS's program table.
[31:24]	Disk address of this input message.
[7:8]	Relative Input queue in which this message resides.

The relative Input queue has a value of 1, 2, or 3 and relates this message to one of three input queues which may be associated with the User program. MSG-RESTART-DATA must be stored in one word of the restart data set.

MSG-SSN (COMMONAREA [3].[39:22]) must also be stored in the restart data set. It contains the system serial number, a unique message number assigned to the message at input and carried with it throughout all phases.

This field must be stored in one word of the restart data set.

The last piece of information to be stored is the Data Base Sequence Number (DBSN), which must be stored in one word of the restart data set. Transactions which have a primary output containing a DBSN of 0 are not recycled during recovery, nor are such transactions archived. This enables the user to intermix update transactions with inquiry transactions in an Input queue declared with RECOVERY = TRUE, without having these inquiry transactions recycled when a recovery cycle is invoked.

A TP, then, must store three words of data in the restart data set for each of three possible Input queues. In addition, all restart areas must be linked with a common key so that the Restart program can retrieve all of the restart data for the recovery mechanism. These considerations point to the restart data set definition for a DMS II data base as shown in figure 5-2. This definition is the one assumed by the COBOL-68 Restart program symbol supplied with the system.

For COBOL-74 programs the fields declared as REAL (S11) in figure 5-2 should be changed to ALPHA (6). This definition is assumed by the COBOL-74 Restart program symbol supplied with the system.

```

RESTARTAREA RESTART DATA SET

  (GEMCOS-LITERAL           ALPHA(6));

  GEMCOS-INTERFACE          GROUP

  (GEMCOS-DATA              REAL (S11));

  GEMCOS-DBSN               REAL (S11);

  GEMCOS-SSN                REAL (S11))
                           OCCURS 3 TIMES);

RESTARTSET SET OF RESTARTAREA

KEY IS GEMCOS-LITERAL DUPLICATES;

```

Figure 5-2. Recommended DMS II Restart Data Set Definition for USER Programs

## Program Interface

A User program that declares RECOVERY = TRUE for one of its queues in the TCL receives two additional parameters from the MCS: a LOCKEVENT and a pointer to the Global DBSN.

In addition, if a User program has WAITFORAUDIT = TRUE declared for one of its queues in the TCL, it receives one additional parameter: a WAITFORAUDIT event.

The WAITFORAUDIT event is used by the User program to coordinate the MCS's auditing of the primary message and the User program waiting for the audit to be completed before leaving DMS II transaction state.

Figures 5-3 and 5-4 show the basic declarations necessary for COBOL and ALGOL User programs with RECOVERY = TRUE specified. Figures 5-5 and 5-6 show the basic declarations necessary for COBOL and ALGOL User programs with RECOVERY = TRUE and WAITFORAUDIT = TRUE specified.

Table 5-1 (following RECOVERY OF A PROCESS PROGRAM) summarizes the various fields mentioned in the preceding recovery discussion and indicates where the fields are located within the COMMON array.

DATA DIVISION.

77 MASTEREVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 SERVICEEVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 EDITOREVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 USEREVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 LOCKEVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 DATABASE-SEQUENCE-NUMBER PIC S9(11) COMP  
RECEIVED BY REFERENCE.

01 COMMON-AREA USAGE IS COMPUTATIONAL  
RECEIVED BY REFERENCE.

02 COMMON PICTURE 9(8) USAGE IS COMPUTATIONAL  
OCCURS 1000 TIMES.

PROCEDURE DIVISION

USING

MASTEREVENT  
SERVICEEVENT  
EDITOREVENT  
USEREVENT  
COMMON-AREA  
LOCKEVENT  
DATABASE-SEQUENCE-NUMBER.

<program>

STOP RUN.

Figure 5-3. COBOL User Program Declarations for a RECOVERY = TRUE  
Specification



```

PROCEDURE USER    (MASTEREVENT,
                   SERVICEEVENT,
                   EDITOREVENT,
                   USEREVENT,
                   COMMON,
                   LOCKEVENT,
                   DBSN);

EVENT            MASTEREVENT,
                   SERVICEEVENT,
                   EDITOREVENT,
                   USEREVENT,
                   LOCKEVENT;

ARRAY            COMMON [0];

REAL             DBSN;

BEGIN

                   <program>

END;

```

Figure 5-4. ALGOL User Program Declarations for a RECOVERY = TRUE Specification

DATA DIVISION.

77 MASTEREVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 SERVICEEVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 EDITOREVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 USEREVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 LOCKEVENT USAGE IS EVENT RECEIVED BY REFERENCE.  
77 DATABASE-SEQUENCE-NUMBER PIC S9(11) COMP RECEIVED  
BY REFERENCE.  
77 WAITFORAUDITEVENT USAGE IS EVENT RECEIVED BY  
REFERENCE.  
  
01 COMMON-AREA USAGE IS COMPUTATIONAL RECEIVED  
BY REFERENCE.  
02 COMMON PICTURE 9(3) USAGE IS COMPUTATIONAL  
OCCURS 1000 TIMES.

PROCEDURE DIVISION

USING

MASTEREVENT  
SERVICEEVENT  
EDITOREVENT  
USEREVENT  
COMMON-AREA  
LOCKEVENT  
DATABASE-SEQUENCE-NUMBER  
WAITFORAUDITEVENT.

<program>

STOP RUN.

Figure 5-5. COBOL User Program Declarations for RECOVERY RECOVERY = TRUE  
and WAITFORAUDIT = TRUE Specification

```

PROCEDURE USER      (MASTEREVENT,
                     SERVICEEVENT,
                     EDITOREVENT,
                     USEREVENT,
                     COMMON,
                     LOCKEVENT,
                     DBSN,
                     WAITFORAUDITEVENT);

EVENT              MASTEREVENT,
                     SERVICEEVENT,
                     EDITOREVENT,
                     USEREVENT,
                     LOCKEVENT,
                     WAITFORAUDITEVENT;

ARRAY COMMON (0);

REAL DBSN;

BEGIN

    <program>

END;

```

Figure 5-6. ALGOL User Program Declarations for RECOVERY = TRUE and  
WAITFORAUDIT = TRUE Specification

## WAITFORAUDIT Program Example

The following is an example of a RECOVERY = TRUE, WAITFORAUDIT = TRUE User program. The program illustrates all of the programming conventions necessary for the MCS to provide a fully synchronized recovery.

Example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE WAITFORAUDIT COBOL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.      86700.
OBJECT-COMPUTER.      86700.
INPUT-OUTPUT SECTION.
DATA DIVISION.
DATA-BASE SECTION.
DB GEMCOSDB ALL.
* 01 GEMCOS-DB STANDARD DATA SET(#2).
*      GEMCOS-DB-SET SET(#3, AUTO) OF GEMCOS-DB KEY IS
*      GEMCOS-VALUE.
* 02 GEMCOS-VALUE PIC 9(8) COMP-2.
* 01 RESTARTAREA RESTART DATA SET (#4).
*      RESTARTSET SET (#5, AUTO) OF RESTARTAREA KEY IS
*      GEMCOS-LITERAL.
* 02 RESTARTTYPE PIC 9 COMP-2.
* 02 TC COMP-4.
* 02 GEMCOS-LITERAL PIC X(6) DISPLAY.
* 02 GEMCOS-INTERFACE OCCURS 3.
*      03 GEMCOS-DATA PIC S9(11) COMP.
*      03 GEMCOS-DBSN PIC S9(11) COMP.
*      03 GEMCOS-SSN PIC S9(11) COMP.
WORKING-STORAGE SECTION.
*****
* THIS IS THE COMMON AREA PASSED FROM THE MCS *
* TO THE USER PROGRAM. IT MUST BE DECLARED *
* AN 01 WITH USAGE = COMP REF. NOTE THAT THE *
* 01 ITEM ITSELF MUST BE COMP, NOT JUST THE *
* INDIVIDUAL ITEMS. *
*****
01 COMMON-AREA                                COMP REF.
   03 COMMON-CONTROL.
       05 WORD-0                                PIC S9(11).
       05 WORD-1                                PIC S9(11).
       05 WORD-2                                PIC S9(11).
       05 WORD-3                                PIC S9(11).
       05 WORD-4                                PIC S9(11).
       05 WORD-5                                PIC S9(11).
       05 WORD-6                                PIC S9(11).
       05 WORD-7                                PIC S9(11).
```

```

05 WORD-8 PIC S9(11).
05 WORD-9 PIC S9(11).
05 WORD-10 PIC S9(11).
05 WORD-11 PIC S9(11).
05 WORD-12 PIC S9(11).
05 WORD-13 PIC S9(11).
05 WORD-14 PIC S9(11).
05 WORD-15 PIC S9(11).
05 WORD-16 PIC S9(11).
05 WORD-17 PIC S9(11).
05 WORD-18 PIC S9(11).
05 WORD-19 PIC S9(11).
05 WORD-20 PIC S9(11).
05 WORD-21 PIC S9(11).
05 WORD-22 PIC S9(11).
03 COMMON-TEXT OCCURS 320 TIMES PIC S9(11).
01 ONE PIC S9(11) COMP VALUE 1.
01 ZERO PIC S9(11) COMP VALUE 0.
01 HIGH-VALUES PIC S9(11) COMP VALUE 65535.
77 EVENTIN EVENT REF.
77 EVENTOUT EVENT REF.
77 SERVICE-EVENT EVENT REF.
77 EDITOR-EVENT EVENT REF.
77 DBSN-EVENT EVENT REF.
77 WAITFORAUDIT-EVENT EVENT REF.
77 DBSN PIC S9(11) COMP REF.
77 AUDITFLAG PIC S9(11) COMP.
77 GEMCOS-IQUINDEX PIC S9(11) COMP VALUE 0.
77 EDT-ABORT-FLAG PIC S9(11) COMP.

```

PROCEDURE DIVISION

```

        USING
            EVENTOUT
            SERVICE-EVENT
            EDITOR-EVENT
            EVENTIN
            COMMON-AREA
            DBSN-EVENT
            DBSN
            WAITFORAUDIT-EVENT.

```

MAIN-CONTROL SECTION.  
100-INITIALIZE.

OPEN SYSTEM FILES AND  
INITIALIZE PROGRAM RESOURCES.

```

OPEN UPDATE GEMCOSDB
ON EXCEPTION
    WAIT AND RESET EVENTIN
    PERFORM DMS-ERROR-ANALYSIS THRU DMS-ERROR-EXIT
    MOVE 2 TO WORD-1
    MOVE 1 TO WORD-0
    CAUSE EVENTOUT
    WAIT AND RESET EVENTIN
    SET MYSELF (TASKVALUE) TO HIGH-VALUES
    STOP RUN

```

ELSE

```

* *****
*
* THE FOLLOWING CODE INITIALIZES
* THE RESTART AREA. THIS MUST BE
* DONE WHENEVER THE DATABASE IS
* OPENED UPDATE.
* *****

CREATE RESTARTAREA
MOVE "GEMCOS" TO GEMCOS-LITERAL
MOVE 0 TO GEMCOS-DBSN(1)
      GEMCOS-DBSN(2)
      GEMCOS-DBSN(3).

200-SLEEP.
    WAIT AND RESET EVENTIN.
    IF MYSELF(TASKVALUE) = 4
        STOP RUN.
    IF EOT-ABORT-FLAG = 1
        MOVE 0 TO EOT-ABORT-FLAG
        MOVE 20 TO WORD-1
        MOVE 1 TO WORD-0
        CAUSE EVENTOUT
        GO TO 200-SLEEP.

300-LOCK-RECORDS.

    LOCK DATABASE RECORDS
    TO BE UPDATED

    MOVE ONE TO WORD-18(0:0:1).
    BEGIN-TRANSACTION NO-AUDIT RESTARTAREA
    ON EXCEPTION
        MOVE ZERO TO WORD-18(0:0:1)
        IF DMSTATUS(ABORT)
            MOVE 20 TO WORD-1
            MOVE 1 TO WORD-0
            CAUSE EVENTOUT
            GO TO 200-SLEEP
        ELSE
            PERFORM DMS-ERROR-ANALYSIS THRU
                DMS-ERROR-EXIT
            GO TO 400-SEND-PRIMARY.

* *****
*
* AFTER THE BEGIN-TRANSACTION, THE FIRST
* THING WE DO IS EXTRACT THE DATABASE SEQUENCE
* NUMBER FOR THIS TRANSACTION, LOCK THE EVENT,
* ADD 1 TO THE PASSED DBSN PARAMETER, AND SAVE
* IT IN WORD 17 OF THE COMMON AREA TO PASS BACK
* TO THE MCS. THEN UNLOCK THE EVENT.
* *****

```

```

LOCK DBSN-EVENT.
ADD 1 TO DBSN.
MOVE DBSN TO WORD-17.
UNLOCK DBSN-EVENT.

```

```

* *****
* *      THIS CODE SETS UP THE RESTART AREA FOR THIS *
* *      TRANSACTION. FIRST WE EXTRACT THE RELATIVE *
* *      QUEUE INDEX FOR THE TRANSACTION TO BE USED *
* *      AS AN INDEX INTO THE RESTART AREA *
* *      GROUP (A GROUP OCCURRING 3 TIMES, ONCE FOR *
* *      EACH OF THE THREE POSSIBLE QUEUES WHICH MAY *
* *      BE ASSOCIATED WITH A USER PROGRAM). SAVE *
* *      THE DBSN JUST ACQUIRED, WORD 10 *
* *      OF THE COMMON AREA, AND THE SYSTEM SERIAL *
* *      NUMBER (WORD [31].[39:22]) IN THE RESTART AREA. *
* *****
*      MOVE WORD-10 TO GEMCOS-IQUINDEX [7:7:8].
*      MOVE WORD-17 TO GEMCOS-DBSN(GEMCOS-IQUINDEX).
*      MOVE WORD-10 TO GEMCOS-DATA(GEMCOS-IQUINDEX).
*      MOVE WORD-3 TO GEMCOS-SSN(GEMCOS-IQUINDEX)
*          [39:21:22].

```

```

UPDATE THE DATABASE.
SEND SECONDARY OUTPUTS.

```

```

FORMULATE THE PRIMARY OUTPUT.

```

```

400-SEND-PRIMARY.
MOVE 3 TO WORD-1.

```

```

* *****
* *      SAVE THE WAITFORAUDIT FLAG FROM THE COMMON *
* *      AREA. THIS INDICATES WHETHER WE HAVE TO WAIT *
* *      FOR THE PRIMARY MESSAGE TO BE AUDITED BEFORE *
* *      PROCEEDING. THIS BIT MUST BE CAPTURED BEFORE *
* *      PASSING CONTROL TO THE MCS WITH THE PRIMARY *
* *      MESSAGE. OTHERWISE THE MCS MIGHT CHANGE THE *
* *      COMMON AREA BEFORE WE GET A CHANCE TO CAPTURE *
* *      THE BIT. *
* *****
*      MOVE WORD-16 TO AUDITFLAG[0:0:1].
*      MOVE 1 TO WORD-0.
*      CAUSE EVENTOUT.

```

```

* *****
* *      WE HAVE JUST SENT OUT THE PRIMARY MESSAGE. *
* *      NOW CHECK IF WE MUST WAIT FOR IT TO BE AUDITED *
* *      BEFORE PROCEEDING. IF YES THEN WAIT ON THE EVENT, *
* *      AND WHEN IT HAPPENS, DON'T FORGET TO RESET IT. *
* *****

```

```

        IF AUDITFLAG EQUAL TO 1
            WAIT AND RESET WAITFORAUDIT-EVENT
*      *****
*      *
*      *   THE PRIMARY OUTPUT HAS BEEN AUDITED.  WE MAY NOW EXIT
*      *   FROM TRANSACTION STATE.  WHEN WE DO, WE CAUSE THE
*      *   RESTART INFORMATION TO BE AUDITED.
*      *****
        IF WORD-18 = 1
            END-TRANSACTION AUDIT RESTARTAREA
            ON EXCEPTION
        ELSE
            MOVE ZERO TO WORD-18(0:0:11).
            GO TO 200-SLEEP.
*****
DMS-ERROR-ANALYSIS
*   DETERMINE TYPE OF DMS II ERROR
*   SET UP ERROR MESSAGE IN COMMON-TEXT AREA
*   TO BE SENT AS A PRIMARY MESSAGE.

DMS-ERROR-EXIT.
EXIT.

```

## The Recovery Cycle

The GEMCOS recovery cycle can be initiated under any of the following conditions:

1. By Network Control command. However, do not initiate recovery by Network Control command unless directed to do so by a system error message.
2. When a User program passes the abort indicator back to the MCS (WORD[1] of the COMMONAREA = 20). Actually, this only causes the MCS to evaluate the status of all User programs and to start recovery for reasons defined in 4 below.
3. When the MCS is restarted after it has abnormally terminated or after the operating system has abnormally terminated (if there was a RECOVERY = TRUE program in the mix at the time of the failure).
4. When a User, Port, or Process program abnormally terminates while in DMS II transaction state.

In the first three cases, recovery is initiated automatically when the situation is recognized by the MCS. In the fourth case, recovery is initiated after the aborted program is disabled and cleared by the operator. If desired, these DISABLE and CLEAR commands could be made automatic by using AUTORECOVERY = TRUE.

The recovery mechanism uses two disk files per system to collect and sort the input transactions to be processed. The names of the two files are SRCFILEnnn and RECFILEnnn (where nnn = the GEMCOS SYSTEM number).

An auditor file is also created by the recovery cycle. The name of this file is EXCEPFILEnnn/hhmm (where nnn = the GEMCOS System number, and hhmm = the creation time in hours and minutes). The auditor records contain control information in the same format as the input and output audit records (refer to appendix B). The user must link to the audit files if the complete text of the records is required (in the case of multiple-record input/output).



The auditor file differs slightly from the format shown in appendix B in the following ways:

Word	Value	Meaning
[INPUTRECORDSIZE]. [13:1]	0	Input Record
	1	Output record
[INPUTRECORDSIZE]. [11:5]	0	Input transaction reprocessed during recovery
	1	Input transaction reprocessed during recovery which had previously caused a TP to abort.
	2	Output record never delivered (has link to corresponding input record within it).

Any transaction which is recycled to a User program during recovery is flagged with the MSG-RECOVERY field (WORD[6].[42:2]) of the COMMONAREA set to one of the following values:

Value	Meaning
0	The system is not in recovery mode.
1	The system is in recovery mode caused by a TP abort.
2	The system is performing an archival recovery.
3	The system is in recovery mode caused by a halt/load or an abnormal termination of the MCS.

In addition, a transaction which causes a User program to abnormally terminate is resubmitted with the MSG-RETRY field in the COMMONAREA (WORD[6].[45:3]) incremented by one. The retry-counter field represents the number of times a transaction was resubmitted for processing, after causing a User program to terminate abnormally. The MCS always resubmits an aborted transaction. The User program must decide how to process the transaction.

During recovery, the system is disabled. When recovery is completed (as reported at the network control stations), the system must be enabled with an ENABLE command. If recovery is initiated by the abnormal termination of a TP, the TP itself must be specifically enabled. These enables could be made automatic by using AUTORECOVERY = TRUE, and the recovery mechanism would be capable of running completely without operator intervention.

Figure 5-7 shows the sequence of events at a Network Control station during recovery after a User program has abnormally terminated in DMS II transaction state. The transaction which caused the program to terminate (DMS1B) is resubmitted during the recovery cycle with the retry-counter field incremented to a value of one. The program action is to ignore the recycled transaction.

Messages preceded by ? are control commands input by the Network Control operator. Messages preceded by \* are messages output by the main MCS stack. Messages preceded by # are messages output by the Network Control stack.

DMS18.12849#228.12\*

\* PROGRAM TERMINATED: 0303:3084 TEST/DMS/PROG/1 ON PACK:  
INPUTQUEUE: DMSQUEUE28(28), NAME: DMTEST10(10).  
MESSAGE ADDRESS:59, MESSAGE #10, STATION: TTY2(1).  
DIVIDE BY ZERO & 00C:04D3:0, 018:0000:5.

\* 0303 RECOVERY CYCLE WILL BE INVOKED FOR SYSTEM (1)  
WHEN PROGRAM (10)-3084 IS DISABLED AND CLEARED

?DISABLE P(10) - 3084

# 0304 PROGRAM (10) DISABLED

?CLEAR P(10)

# 0304 PROGRAM (10) CLEARED

\* 0304 RECOVERY CYCLE INITIATED FOR SYSTEM (1)

\* 0304 RESTART TP INVOKED FOR SYSTEM (1)

\* 0304 RECOVERY STACK INVOKED FOR SYSTEM (1)

\* 0305 RECOVERY STACK COMPLETED FOR SYSTEM (1)

DMS18.12849#228.12\* TRANSACTION IGNORED

\* 0306 RESTART TP COMPLETED FOR SYSTEM (1)

\* 0306 SYSTEM (1) RECOVERED

?ENABLE ALL

# 0306 MGTESTSYS1 (1) ALL ENABLED

?ENABLE P(10)

# 0307 PROGRAM (1) ENABLED

Figure 5-7. User Program - Abnormal Termination Recovery Cycle

## Recovery of a Process Program

The programming conventions necessary for the recovery of Process programs are the same as for User programs. The difference is that the various pieces of control information necessary to satisfy these conventions are found in the CONTROLWORDS array passed to the Process program rather than in the COMMONAREA passed to a User program.

Similar to User programs, Process programs receive additional parameters to coordinate the recovery process. A RECOVERY = TRUE Process program receives the LOCKEVENT and a pointer to the Global DBSN. A RECOVERY = TRUE, WAITFORAUDIT = TRUE Process program also receives the WAITFORAUDITEVENT. See figures 5-8 and 5-9 for the basic declarations of a RECOVERY = TRUE and a RECOVERY = TRUE, WAITFORAUDIT = TRUE Process program.

Table 5-1 summarizes the various fields mentioned in the recovery discussion of the preceding pages and indicates where the fields are located within the CONTROLWORDS array.

```
PROCEDURE PROCESSPROG      (CONTROLWORDS,  
                             MASTEREVENT,  
                             PROCESSEVENT,  
                             GETMESSAGE,  
                             SENDMESSAGE,  
                             LOCKEVENT,  
                             DBSN);  
  
ARRAY CONTROLWORDS(0);  
  
EVENT  MASTEREVENT,  
        PROCESSEVENT,  
        LOCKEVENT;  
  
PROCEDURE GETMESSAGE,  
          SENDMESSAGE;  
REAL    DBSN;  
  
BEGIN  
  
    <program>  
  
END;
```

Figure 5-8. Process Program Declarations for a RECOVERY = TRUE Specification

```

PROCEDURE PROCESSPROG      (CONTROLWORDS,
                             MASTEREVENT,
                             PROCESSEVENT,
                             GETMESSAGE,
                             SENDMESSAGE,
                             LOCKEVENT,
                             DBSN,
                             WAITFORAUDITEVENT);

    ARRAY CONTROLWORDS[0];

    EVENT MASTEREVENT,
    PROCESSEVENT,
    LOCKEVENT,
    WAITFORAUDITEVENT;

    PROCEDURE GETMESSAGE,
    SENDMESSAGE;
    REAL    DBSN;

    BEGIN

        <program>

    END;

```

Figure 5-9. Process Program Declarations for a RECOVERY = TRUE, WAITFORAUDIT = TRUE Specification

The transaction conventions within the Process program relative to record locking, BEGIN and END transactions, DBSN retrieval, restart information store, and WAITFORAUDIT should be the same as described for User-type programs. When an abort exception is detected on a DMSII BEGIN transaction, the program should reset the transaction state bit, place 20 into the Task-value, and cause the MASTEREVENT. When an abort exception is detected on a DMSII END transaction, the program, upon receiving the next input (PROCESSEVENT), should place 20 in the Task-value and cause the MASTEREVENT. The Process program should only make one primary-type SENDMESSAGE call, and it should be the last SENDMESSAGE call, before setting its Task-value to 3 and causing the MASTEREVENT. A WAITFORAUDIT = TRUE Process program should wait for its WAITFORAUDIT-EVENT after setting its Task-value to 3 and causing the MASTEREVENT.

Table 5-1. Summary of Recovery Fields

Field Description	Field in COMMONAREA	Field in CONTROLWORDS ARRAY
System serial number	MSG-SSN ([3].[39:22])	[4].[23:24]
Transaction retry-counter	MSG-RETRY ([6].[45:3])	[4].[37:3]
Transaction is being reprocessed by recovery.	MSG-RECOVERY ([6].[42:2])	[16].[47:2]
Input queue number as defined in the TCL in which this message resides.	[10].[47:8]	[4].[32:8]
Index into MCS's program table	[10].[39:8]	[16].[39:8]
Disk address of input record	[10].[31:24]	[16].[23:24]
Relative Input queue in which this message resides	MSG-REL-IQU ([10].[7:8])	[4].[34:2]
WAITFORAUDIT bit	MSG-WAITAUDIT ([16].[0:11])	[16].[42:1]
DBSN passed back to MCS to stamp output messages.	MSG-DBSN ([17])	[18].[39:39]
TRANSACTIONSTATE bit (Port programs must also set TASKVALUE to -1)	MSG-TRANSTATE ([18].[0:1])	[16].[41:1]

## Output Message Analysis

Output to the network may be generated by transactions which are recycled during recovery. In order to maintain the integrity of messages received at terminals and attempt to avoid duplication, the MCS does an extensive analysis of output messages generated during recovery.

If the messages generated by the original processing of the transaction were not released to the network prior to the fault, the old messages are deleted and the new messages are sent out. If the old messages were released to the network prior to the fault, the new messages generated during recovery are not sent.

There are circumstances during the recovery cycle when it is not possible to determine if a particular message was received at a terminal or not. The MCS realizes that if it releases the message during recovery, it might be releasing a duplicate message. If NODUPLICATES = TRUE is specified for a terminal, possible duplicate messages are not released. Instead, a warning message is sent to the terminal, stating that a possible duplicate message is being held up. Along with the warning, instructions are provided on how to recall the message using the REFRESH Network Control command.

In order to reduce the maximum number of such possible duplicates to one for a given station, the following TCL syntax may be specified in the STATION section:

ONEOUTPUTPERBACKUP = TRUE.

This prevents the next output for this station from being released until the acknowledgement of the last output is audited by the MCS.

## Archival Recovery

There are instances when it might become necessary to reconstruct the sequence of events performed upon the data as initiated by the data communications network over a period of many days. Conditions necessitating such action might arise if DMS II reconstruction should fail, or if a TP bug has contaminated the data base beginning several days ago. The archival recovery mechanism is the means by which to accomplish this reconstruction of events.

Archival recovery uses the archival audit dumps created each time the Transaction Control files are generated or regenerated, or when a ?DUMP DATACOM Network Control command is entered in a CONTINUOUSPROCESSING environment. Further information is presented under the ARCHIVALAUDIT statement syntax contained in section 3. The file description of the archive tape is given in appendix B. To initiate an archival recovery, the program GEMCOS/ARCHREC should be executed with all system files (input/output, control, format, and ledger) label-equated as needed. Certain controlling information must be supplied either by card or through the display console, beginning with a card file verification check to determine if the file is present. The acceptable card file is CARD.

The following information must be supplied:

1. The systems to be recovered. (One system, many systems, or ALL systems may be specified.)
2. The date on which the audit files were (re)generated.
3. The sequence control number of the files on the archival dump. (This is the same number that appeared on the sequence control statement when the files were originally generated.)
4. The name of the MCS, if other than GEMCOS/MCS.
5. The date and time that the archive tape was created (date and time used in tape name).

This information must be supplied in free format according to the following syntax:

```
<archival recovery request> ::=
    RECOVER <system specification>
    <gen dump specification>
    <MCS title specification>

<system specification> ::=
    SYSTEMS <optional equal> (<system list>)

<gen dump specification> ::=
    <gendate specification>
    <sequence control specification> /
    <dump date specification> <dump time specification>
```

```

<optional equal> ::= = / <empty>

<system list> ::= ALL / <system specification list>

<system specification list> ::=
    <system number> /
    <system specification list>, <system number>

<system number> ::= <integer>

<gendate specification> ::= GENDATE <optional equal> <date>

<date> ::= <month> <slash> <day> <slash> <year>

<month> ::= <integer>

<day> ::= <integer>

<year> ::= <integer>

<sequence control specification> ::=
    SC <optional equal> <integer>

<dump date specification> ::=
    DUMPDATE <optional equal> <date>

<dump time specification> ::=
    DUMPTIME <optional equal> <integer>

<MCS title specification> ::=
    MCSTITLE <optional equal> [a valid MCS title] / <empty>

```

Before initiating the archival recovery mechanism, the operator must ensure that the data base is loaded from a dump tape that reflects the state of the data base at the START of the archival dump.

The GEMCOS/ARCHREC program scans the ledger file to determine which archival dump tapes to use during the recovery cycle. It then reads the archival tape, writes the files to disk, and proceeds to RUN the MCS to initiate recovery. Messages are printed at Network Control stations to indicate the status of the recovery cycle.

The Input queues can be selectively recovered. The recovery stack first awaits a response to an ACCEPT for specification of nonrecoverable queues to retrieve from audit tapes.

Example:

```

DISPLAY: ENTER NONRECOVERABLE IQUS
ACCEPT: ALL, NONE, IQ#, OR IQ NAME

```

When an individual Input queue name or number is specified, the ACCEPT is repeated as below until END is specified.

ACCEPT: END, IQ#, or IQ NAME

The ACCEPT cycle is then repeated for recoverable Input queues.

If there are valid messages in the current MCS files at the time it becomes necessary to initiate the archival recovery mechanism, the files should be regenerated as the next generation of MCS files, in order to create an archival dump of these current files. This new archival dump can then be processed as part of the recovery mechanism.

The archival recovery mechanism recovers only one generation of TCL files, i.e., one archival dump. Queues are built and the MCS is processed. When the MCS goes to EOJ, if additional archival tape information is contained in the ledger file, the operator has the option of recovering the next generation of TCL files. Otherwise, the archival recovery program GEMCOS/ARCH-REC must be run again with parameters for the next archival dump.

During archival recovery, no messages generated by TPs are released to terminals except those whose destination is specifically the System Network Control station.

## Recovery of a Non-DBMS Transaction

If a TP terminates abnormally while not in transaction state, the recovery mechanism is not initiated. When the MCS recognizes that a TP has terminated outside of its control (i.e., abnormal program termination, premature termination by an operator, or a "normal" program termination without being told by the MCS), it sends a system error message to the Network Control stations. That program is then inoperative until the operator takes appropriate action.

The operator must DISABLE, CLEAR, and then ENABLE the aborted TP. This activates the program and resubmits the transaction which caused it to abort with the retry-counter field incremented by one. If desired, the DISABLE, CLEAR, and ENABLE commands could be made automatic by specifying AUTORECOVERY = TRUE in the System section.

## Synchronized Recovery of Batch Jobs

Before opening the remote file BATCHLSN, a Batch job should check the value of its RESTARTED task attribute. A restarted Batch job (i.e., MYSELF(RESTARTED) = TRUE) must wait on some form of operator intervention before opening the remote file. The operator can allow a restarted Batch job to continue when the MCS has initialized the Primary queue and all its stacks are running. This procedure must be followed to prevent duplicate MCS stacks from entering the mix after a Halt Load.

If the Batch job's handshake is accepted by the MCS, the MCS returns a computational number in the first word of the response. This number is the transaction count of the last transaction received (as indicated by the first four characters of the original transaction initiated by the Batch job) and audited by the MCS from this Batch job. Since this transaction count is typically a pointer into a serial file being processed by the Batch job, a nonzero count returned would indicate the number of records of the serial file that must be skipped before the batch job can continue processing the file.



During recovery, all stations are disabled, including OBJECTSTA, the station through which Batch jobs correspond with the MCS.

#### NOTE

OBJECTSTA does not appear in either the NDL or the TCL specifications.

An ENABLE ALL Network Control command does not enable OBJECTSTA. It must be done explicitly by specifying ENABLE S OBJECTSTA when OBJECTSTA has output queued for it. In order not to confuse the response to the handshake with the response to an input entered prior to the failure, OBJECTSTA must not be enabled until all Batch jobs have received their responses to the handshake message. This means that a Batch job must wait for some form of operator intervention if it receives a nonzero response to its handshake. The operator can allow all Batch jobs to continue after they have all received responses to their handshakes (i.e., when they are all waiting).

There may be many output messages queued for a Batch job during the recovery cycle. These messages comprise the responses to the last transaction audited by the MCS prior to the failure. In order to cause the Batch job to synchronize any subsequent input with its corresponding output, the user may wish to transmit a special "marker" transaction. This causes all the responses queued for the Batch job to be read until the response to the marker transaction is encountered. All subsequent responses received by the Batch job can then be properly associated with their corresponding input.

A detailed discussion of the Batch Job Interface is provided in Section 4.

## Synchronized Recovery of TP-to-TP Transactions

Two types of TP-to-TP transactions are possible. In type I TP-to-TP transactions, the originating TP continues normal processing after emitting the transaction and does not wait for a response from the destination TP. All responses from the destination TP are routed normally, including the primary output which is routed to the station which entered the original transaction. Even though the individual type I transactions are recoverable, a sequence of such transactions from a TP is not recoverable, and duplicates may result if a synchronized recovery is required. Type I transactions are indicated by placing a 2 in MSG-DESTTYPE (COMMON[4].[39:8]).

In type II TP-to-TP transactions, the primary output of the destination TP is directed back to the originating TP. These type II TP-to-TP transactions are addressed by the GEMCOS synchronized recovery mechanism, and recovery is transparent to the secondary TPs.

The following must be performed by the originating TP to invoke type II TP-to-TP transactions:

1. Place the message into the text section of the COMMON AREA.
2. Place value 2 into MSG-ACTION ([COMMON[1].[23:24]]) to indicate the end of the current message to destination.

#### NOTE

The message may be released to the MCS in segments by using value 0 for the first segments and value 2 for the last.

3. Place value 7 in MSG-DESTTYPE (COMMON[4].[39:8]) to indicate that the destination is a TP and that the primary response should be directed back to the originating program.
4. Place value 0 in MSG-DEST (COMMON[4].[19:20]) to indicate routing of the message is to be by the message key which is part of the message.
5. Place value 1 in MSG-CONTROL (COMMON[0]) and cause the MCS event.

Messages directed to a TP using this mechanism are given priority over messages directed to the TP from a terminal via the standard queueing mechanism. All secondary messages sent out by the destination TP are routed normally to stations, areas, etc. No secondary messages can be routed back to the originating TP. A primary output whose destination is the originator is delivered to the originating TP. If the primary output is not directed to the originator, a dummy output of length 0 is sent to the originating TP.

The normal MCS synchronized recovery mechanism is capable of handling TP-to-TP messages which are larger than the destination TP's Common row. The response sent back to the originating TP, however, must not be larger than the originating TP's Common row. A TP may not direct a message to itself. The destination TP may not send out a TP-to-TP message other than a response message to the originator.

After sending out a TP-to-TP message as indicated above, the originating TP waits on its input event for the response. If the response is normal, value 6 is returned in MSG-ACTION (COMMON[1]), the response is returned in the text portion of COMMON, and the length of the response is returned in MSG-LENGTHIN (COMMON[7]).

If the response is not normal, value 5 is returned in MSG-ACTION (COMMON [1]), and a result indicator is returned in MSG-LENGTHIN (COMMON[7]):

Value	Meaning
137	Indicates either that the destination program was inoperative (NOP) or that the destination program abnormally terminated while processing the message.
138	Indicates that the response of the destination TP was larger than the Common row of the originating TP.

If the TP-to-TP messages cause data base updating, the following conventions are required to allow these messages to be properly handled by the recovery mechanism.

1. The originating TP must lock all data base resources which might be updated while the TP-to-TP dialogue is taking place. The user might want to use a "soft-lock" on some critical node (e.g., an account number), which would prevent accessing of records lower in the hierarchy. The use of a "real" DMS II lock on a critical node could cause contention among the originating and destination TPs.
2. If the originating TP has to enter DMS II transaction state to set the soft lock, it should not acquire a DBSN. It must then leave transaction state before directing messages to other TPs. When leaving transaction state, the restart data should NOT be written to the DMS II audit trail.
3. When all TP-to-TP messages are sent and responded to, the originating TP should enter DMS II transaction state, obtain a DBSN, remove the soft-lock if one was necessary, generate the primary output, update the restart set work area, and leave transaction state. This causes the restart data to be written onto the DMS II audit trail.

Each destination TP may process its transaction exactly the same as if it came from a terminal instead of another TP. If the destination TP must know who the originator is, it can check MSG-MSGTYPE (COMMON[3].[42:3]). The MCS places one of the following values in this field to indicate the originator of the message:

Value	Meaning
0	A normal station message.
1	The message was generated by a program, but the response is sent to station at which the original input message was entered.
2	The message is a system-defined input message.
3	The message was generated by a program, and the response will go back to the originating program.

During a recovery involving TP-to-TP transactions, there is no convenient way for the originating TP to determine which subtransactions are reflected on the data base within a given total input transaction currently being recycled. This is not a major problem in situations where recovery was NOT caused by abnormal termination of one of the destination TPs that was involved in the processing of the overall transaction. In such situations, the retry counter for the input transaction from the network remains set at zero. The input transaction can basically be processed as though it was never seen before (although the application may ignore any soft locks for the transaction that may still be on). GEMCOS performs an output analysis during recovery such that any subtransaction (secondary input) not rolled off the data base and resubmitted by the originating TP, is not redelivered to the destination TP. Instead, the corresponding original response of the destination TP is extracted from the Output Audit queue for redelivery to the originating TP. This analysis is based upon the DBSNs contained in the previous response to the recycled subtransactions. Thus, destination TPs should WAITFORAUDIT when issuing responses.

There is a potential problem when the retry counter for the original input from the network is incremented as a result of the abnormal termination of one of the destination TPs in the TP-to-TP sequence. During recovery, the originating TP must first make the decision of whether to retry the transaction sequence, which could result in another recovery. If processing of the transaction must be bypassed, the original transaction might be partially reflected on the data base. Additionally, GEMCOS still expects a primary output to be generated to the originating station for that input to clear Transaction mode. If the transaction must be bypassed, the originating TP must notify someone of the problem so that some sort of exception-handling can be invoked.

## Transaction Processing System (TPS) Recovery

This type of recovery is used in a Transaction Processing System (TPS) environment. A system may have either TPS recovery or synchronized recovery specified, but not both.

Input queue pointers are positioned to the first input (after the last checkpoint position) which does not have a primary output message being audited. All inputs without outputs are resubmitted to the programs. The first message resubmitted for each Input queue is marked in a special way. It is then the responsibility of the program to check whether this message has already been applied to the data base. (Refer to SAMPLE TPS PROGRAM below for more detail.)

## TCL Requirements

The following TCL options should be set:

1. RECOVERY = FALSE.
2. WAITFORAUDIT = TRUE.

## TPS Programming Conventions

The following are the conventions for TPS programming:

1. Each input message should have a unique identifier. This identifier should be saved in the Output Transaction record. It is recommended that the COMMONAREA field MSG-SSN (WORD[3].[39:22]) be used.
2. After sending a primary output and causing the MCS event, the program may wait on the AUDITEVENT, but it is not necessary to do so. In either case, the MCS ensures that the output gets audited before the program is given a new input message.
3. If a program has more than one Input queue defined, it should log on more TRUSERS (call on LOGONTRUSERS). The TCL queue-ID could be used as the TRUSER-ID string. Each call on PROCESSTRNORESTART should use the ID-number associated with the appropriate queue. During recovery, the program has access to the last response for each queue.
4. To execute a transaction the program can call on PROCESSTRNORESTART. During recovery, we can retrieve the last output transaction record with a call on RETURN-LASTRESPONSE. The input message number can be compared with the number saved in the Output Transaction record to determine whether this transaction has already been applied to the data base. If it has, the output needs only to be sent back to the user. Otherwise, this message should be reprocessed.
5. Please note that multiple copies are disallowed in conjunction with TPS at the present time.
6. During a recovery following a halt/load, the MCS resubmits all input messages that had no primary output. The first message of each queue has a value of 3 in MSG-RECOVERY (WORD[6].[42:2]) of its COMMONAREA. This is to signal to the TP that this input does not have an output but that the transaction could have been applied to the data base. It is up to the TP to decide whether to resubmit the input to the TPS or merely to send the output to the user.
7. The same applies to a TP that died while processing a transaction. Its Input queues are marked and a value of 3 appears in the COMMONAREA field MSG-RECOVERY (WORD[6].[42:2]).

## Sample TPS Program:

```
IDENTIFICATION DIVISION.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
DATA-BASE SECTION.  
DB      MYDB ALL.  
TRANSACTION SECTION.  
TB      MYTRB.  
01  TR-IN.  
01  TR-OUT.  
WORKING-STORAGE SECTION.
```

```

.
.
PROCEDURE DIVISION USING . . . . .
INITIALIZE.
    CALL "OPENTRBASE OF MYTRB/CODE/HOSTLIB" USING
        UPD TIMEOUT GIVING ERR-INDICATOR.
    IF ERR-INDICATOR GTR 0 .....
        MOVE PROGRAM-NO OF GEMCOS-COM TO ID-STR1.
        MOVE INPUTQ-REL OF GEMCOS-COM TO ID-STR2.
        CALL "LOGONTRUSER OF MYTRB/CODE/HOSTLIB USING
            ID-STR ID-NUM GIVING ERR-INDICATOR.
        IF ERR-INDICATOR GTR 0 .....4...
MESSAGE-LOOP.
    WAIT AND RESET EVENT-IN.
    IF MYSELF (TASKVALUE) = 4 GO TO EOJ.
    MOVE GEMCOS-RETRY TO TEMP-RETRY (45:2:3).
    IF TEMP-RETRY GTR 0
        MOVE "ERROR IN TRANSACTION" TO GEMCOS-TEXT
        MOVE 20 TO GEMCOS-LENGTH
        GO TO SEND-IT.
    MOVE GEMCOS-RECOVERY TO TEMP-RECOVERY (42:1:2).
    IF TEMP-RECOVERY = 3
        PERFORM CHECK-RECOVERY THRU CHECK-RECOVERY-EXIT.
    IF SKIP = 1
        MOVE 0 TO SKIP
        GO TO BUILD-OUTPUT.
    PERFORM PROCESS-INPUT THRU PROCESS-INPUT-EXIT.
BUILD-OUTPUT.
    MOVE CORRESPONDING OUTFORMAT OF TR-OUT TO GEMCOS-TEXT.
    MOVE OUT-LENGTH TO GEMCOS-LENGTH.
SEND-IT.
    MOVE 3 TO GEMCOS-ACTION.
    MOVE 1 TO GEMCOS-CONTROL.
    CAUSE EVENT-OUT.
    GO TO MESSAGE-LOOP.
EOJ.
    STOP RUN.

CHECK-RECOVERY.

    CALL "RETURNLASTRESPONSE OF MYTRB/CODE/HOSTLIB" USING
        ID-NUM TR-OUT GIVING ERR-INDICATOR.
    IF ERR-INDICATOR .....
        MOVE GEMCOS-SSN TO TEMP-SSN (39:21:22).
        IF TEMP-SSN = SSN OF OUTFORMAT OF TR-OUT
            MOVE 1 TO SKIP.
CHECK-RECOVERY-EXIT.
EXIT.

```

```

PROCESS-INPUT.
  CREATE INFORMAT OF TR-IN.
  MOVE CORRESPONDING GEMCOS-TEXT TO INFORMAT OF TR-IN.
  MOVE GEMCOS-SSN TO TEMP-SSN (39:21:22).
  MOVE TEMP-SSN TO SSN OF INFORMAT OF TR-IN.
  CALL "PROCESSTNORESTART OF MYTRB/CODE/HOSTLIB" USING
    ID-NUM TR-IN TR-OUT GIVING ERR-INDICATOR.
  IF ERR-INDICATOR .....
PROCESS-INPUT-EXIT.
  EXIT.

```

## Summary

GEMCOS plays the major role in effectively recovering from system failures and in maintaining data base and network integrity in the process. GEMCOS provides a wide range of recovery capabilities, freeing the user from recovery concerns to the maximum extent possible. The high-level flexibility offered by the system allows the user to analyze application requirements and then select the recovery options best suited to meet specific data processing needs.



## SECTION 6

# FORMATTING AND PAGING

The Transaction Control Language (TCL) section describes the syntax and semantics of the language constructs associated with formatting (refer to section 3).

The Message Control System (MCS) retrieves format descriptions based on both message-identification key and the device class of the station involved and applies the format to the data. This allows tremendous flexibility at the stations, transparent to the application programmer. For example, the application program can tell the MCS to output a given message to two stations on the same network. If those two stations are described as being in different device classes, the resulting output can be drastically different based upon two different format descriptions provided in the TCL. Such data-field characteristics as length, sequence, form information, and whether certain fields are required or optional, can vary from station to station. Conversely, two different stations can supply input for the same type of transaction under different formats, and the data arrives at the application program in a standard format. In fact, in cases where each input field from a terminal is delimited by a unique key, the sequence of data entry can vary from transaction to transaction at the same station, even though the same type of transaction is being repetitively performed. Such activity is transparent to the application program under the formatting feature of GEMCOS.

## INPUT FORMATTING

Formatting of an input message is accomplished if and only if:

1. An Editor program is defined for the destination User program.
2. The message key of the input transaction is associated with a particular format in the <device section> of the TCL for the device containing the originating station.

The Editor receives its input through the Editor queues. The standard Editor supplied with GEMCOS is designed to do message formatting and to insert the edited message into the COMMONAREA for the User program to process. The standard Editor sets up COMMONAREA, just as the MCS would do if no editor were being used, i.e., the control-word MSG-CONTROL (COMMON[1]), the action word MSG-ACTION (COMMON[2]) and the message-length-word MSG-LENGTHIN (COMMON[7]) contain values just as if the MCS itself had passed control directly to the User program. (Conventions are different for Editors of Port type programs).

If a nonstandard Editor program is used (i.e., one written by the user), any interface connection between the Editor and User program may be used. For instance, the User program may give control back to the Editor program (by causing its event) as long as the Editor was programmed in this case to look for the message in the COMMONAREA. If it is necessary for the User program to know the length of the input message, then it is the Editor's responsibility to ensure that the length of the message passed to the User program is correctly specified in MSG-LENGTHIN (WORD[7] of the Common Area).



## OUTPUT FORMATTING

Formatting of an output message is accomplished if and only if the following two conditions are met:

1. There is an association between an output message-ID and the desired format specified in the <device section> of the TCL for the device containing the destination station.
2. A TP places the output message-ID into MSG-MESSAGE-ID (word [9] of the Common area).

The format is retrieved and processed with the appropriate data from the message inserted as required by the format. The resulting message is released to the terminal.

## FORMS REQUEST

To request a blank form, a station transmits an output message-ID with no data, including no special terminating character, as required at a variable length message-key station. The following is an example of an unpagged forms request invoked at a terminal.

Example:

DEFINE

```
FF = 4"OC"#,
CR = 4"OD"#,
LF = 4"25"#,
FORM = 4"12"#,
TAB = 4"05"#.
```

```
FORMAT F1 (FF, "[MKE1.]", X1,
"INSERT QUANTITY [", I5, "]",
CR, LF, X8,
"INSERT TYPE [", A3, "]",
FORM, TAB).
```

```
DEVICE TD800(1):
  STALIST = TD69.
  FORMATSOUT:
  F1 = FM100.
```

The forms request as it would appear on the screen is:

FM100

The result of the forms request as it would appear on the screen is:

```
[MKE1.] INSERT QUANTITY [ ]
  INSERT TYPE [ ]
```

To invoke unpagged output editing, the following message is placed in the text portion of the message by a TP:

12345BT1

I5 A3

with "FM100 " in MSG-MESSAGE-ID (word[9] of the COMMONAREA).

The message as it would appear at the terminal is:

```
[MKE1.] INSERT QUANTITY [12345]
      INSERT TYPE [BT1]
```

## PAGED FORMATS

If an output format is paged, certain additional restrictions and conventions must be followed. When a paged output format is to be applied to a message before it is sent to a terminal, the terminal enters into paging mode. Each page of the format is applied to the appropriate part of the message and sent to the terminal at the request of the terminal operator. The terminal operator may browse at will through the output pages.

### Display Paging

Display paging allows the terminal operator to browse through output pages as described above and review the requested data. While in display paging mode, upon each transmission of data from the terminal, the MCS looks only at the first character of the data transmitted, takes the appropriate action, and discards the remainder of the transmitted message, if any. The following is an example of a display paging dialogue.

Example:

```
DEFINE
  HOME = 4"3C"#,
  FF = 4"0C"#,
  CR = 4"0D"#,
  LF = 4"25"#,
  FORM = 4"12"#,
  TAB = 4"05"#.

FORMAT F2 (PAGE[1]:
  FF, " + PART NAME ", A10,
  CR, LF, X2,
  "PART NUMBER ", I5,
  HOME, " + ",

PAGE[2]:
  FF, " + NUMBER IN STOCK ", I5,
  CR, LF, X2,
  "REORDER NUMBER ", I5,
  HOME, " + ").
```

The message placed in the text portion of the message by a TP is:

BIG WIDGET039870010000100

The message as it would appear on the screen for pages 1 and 2 is:

Page 1	Page 2
+ PART NAME BIG WIDGET	+ NUMBER IN STOCK 00100
PART NUMBER 03987	REORDER NUMBER 00100

## Update Paging

In addition to enabling the terminal operator to browse through pages to review data, update paging also allows updating of any protected data as the browsing is done. As each page is displayed on the screen, the operator may insert or change information in the unprotected fields of the format. Each time the operator transmits a paging request, the current screen is transmitted as well and is used by GEMCOS to update the raw message accordingly. If that page is recalled to the screen at a later time in the paging dialogue, the newly updated information is displayed in the form. When finished inserting or updating data, the terminal operator can discontinue paging and transmit the entire message to a TP.

The following example illustrates an update message format and its effect on output and input messages. The + and X which appear in this example are paging dialogue characters, explained subsequently. When the operator has finished updating fields, only the operator-accessible (unprotected fields) are transmitted. This data is used to update the original message. In this example, note the use of the L <item type> following the <editing string> within an unprotected field. This feature is very useful when it is desirable to have the update format insert a message key into the updated message.

Example:

```
FORMAT F2 [UPDATE]
(PAGE[1]:FF "[+MKE2.] ", L5,
" PRODUCT: " , C7, X4,
" STOCK# " , C5, CR, LF, X18,
" REORDER POINT [" , I4, "]" ,
FORM, TAB,
PAGE[2]:FF, "[X] NUMBER ON ORDER [" ,
I4, "]" , CR, LF, X4,
"EXPECTED DELIVERY [" ,I6,"]" ,
FORM, TAB).
```

The message placed in the text area of COMMONAREA by the User program is then:

00000WIDGETS7641201002500010180

L5 C7 C5 I4 I4 I6

Page 1, as it would appear on the screen after the previous message is formatted, is as follows:

```
[+MKE2.] PRODUCT: WIDGETS STOCK# 76412  
REORDER POINT [0100]
```

The operator updates the REORDER POINT field, which changes the screen to:

```
[+MKE2.] PRODUCT: WIDGETS STOCK# 76412  
REORDER POINT [0150]
```

When the page is transmitted, the following data is returned to the MCS:

```
+MKE2.0150
```

The raw message after the update is completed would appear as:

```
MKE2.WIDGETS76541201502500010180
```

Page 2 then appears on the screen as follows:

```
[X] NUMBER ON ORDER [2500]  
EXPECTED DELIVERY [010180]
```

The operator updates the NUMBER ON ORDER and EXPECTED DELIVERY fields which changes the screen to:

```
[X] NUMBER ON ORDER [1550]  
EXPECTED DELIVERY [040580]
```

When the page is transmitted, the following data is returned to the MCS:

```
X1550040580
```

The raw message after the update is completed would appear as:

```
MKE2.WIDGETS7641201501550040580
```

## The Paging Dialogue

A station enters paging mode whenever an output to that station involves a paged format. In paging mode, the station operator enters into a dialogue with the MCS and may browse through all pages of the message.

The dialogue between the station and the MCS in paging mode keys off the first character of the data transmitted. The following are the characters the MCS looks for in paging mode and the action called for by the MCS upon receipt of each character:

Character	MCS Interpretation
+	Display the next page in sequence. When received on the last page, wrap around to the first page.
-	Display the previous page. When received on the first page and the last page was already seen, wrap around to the last page. If the last page has not yet been seen, this is invalid on the first page.
C	Exit from paging mode. Clear the screen and all pages in memory. The message CANCELLED is displayed at the terminal.
X	Discontinue paging and transmit the entire message to the TP (illegal in display paging).
R	Refresh the screen with the most current page.
H	Return the first page to the screen (analogous to) the home key on the keyboard).
D	Return to the beginning of the current paging dialogue. For display paging, this is similar to the H command. For update paging, this causes the accumulated message to be discarded and the original paging dialogue restarted from the first page. On a forms request, this is similar to the C command.
?	(Where ? is any other character). Exit paging mod. Valid only in display paging when 1) the last page of the message has been sent or 2) message size is greater than 1 and the first character is not a "+" or "-". The input is treated as a new message by the MCS. In all other cases an unrecognizable character (not one of the above) is treated as an error; the station remains in paging mode.

In display paging, if the size of the input is larger than that of the dialogue character and the first character is not "+" or "-", then the input is treated as a message key. Paging mode is exited and the input is forwarded to the MCS.

## FORMAT LIBRARIES

The GEMCOS Format Library interface provides a user the ability to write message formatting code in ALGOL, COBOL, or COBOL74, in addition to the standard TCL formats. Message formatting using a high level language not only allows a user to enhance formatting capabilities on his own as new problems arise, but also decreases the system resources required to format any message. Along with the greater flexibility provided by this interface, the library writer picks up new responsibilities; i.e., the writer must follow the format library conventions carefully. However, sample ALGOL and COBOL libraries are provided with the GEMCOS release, and should ease the transition from TCL to library formatting.

A format library must exist in a code file titled `"*GEMCOS/FORMATLIBRARY"` (note: there is no usercode); it has one entry point, `FORMATLIB`, and four parameters, `TERMMSG`, `MCSMSG`, `PAGERTABLE`, and `MISCPARAM`. `MISCPARAM` is further divided into eight six-character fields, the first of which selects the function the library is to perform on this call. These functions are:

#### 1. LIBRARY INITIALIZATION

During GEMCOS initialization, the library provides information to GEMCOS concerning which formats are available in the library, and the attributes of each of these formats.

#### 2. INPUT FORMATTING

Given a message from a terminal, format it for delivery to a TP.

#### 3. OUTPUT FORMATTING

Given a message from a TP, format it for transmission to a terminal.

#### 4. PAGED FORMAT INITIALIZATION

Paged formatting implies a multiple screen dialogue in which GEMCOS displays a screen, the user modifies portions of the screen and transmits a message back to GEMCOS. Part of this message is used to determine which page should be the next displayed. GEMCOS is responsible for storing relevant data (which may be changed by the format library) over the course of the dialogue and delivering this data to a TP upon completion of the dialogue.

The format library is given an array where it can build a table of dialogue related information, such as the current page number, or a list of page indices, but the definition of the data in this table and how that information is used is left to the library developer. Paged format initialization is responsible for setting up this table for a subsequent paged output call on the library.

#### 5. PAGED INPUT FORMATTING

Given the stored message, `PAGERTABLE` and a message from a terminal, update the stored message and `PAGERTABLE` as necessary, and set `PAGEFUNC` to request appropriate action of GEMCOS. One possible result is to transmit a page to the terminal, which required that `PAGERTABLE` be set up for a call on paged output to format that page.

#### 6. PAGED OUTPUT FORMATTING

Given the stored message, and `PAGERTABLE`, prepare the appropriate screen to be sent to the terminal.

#### 7. PAGED RECALL FORMATTING

After setting up `PAGERTABLE` by a call on paged initialization, paged recall is called repeatedly until each page has been formatted. The library is responsible for informing GEMCOS when all the pages of the message have been formatted.

## Library Parameters

Four parameters are passed to the library: TERMMSG, MCSMSG, PAGERTABLE, and MISCPARAM. The following table shows the type of each parameter for the format library implementation languages:

	ALGOL	COBOL	COBOL74
TERMMSG	EBCDIC ARRAY	01 DISPLAY	01 DISPLAY
MCSMSG	EBCDIC ARRAY	01 DISPLAY	01 DISPLAY
PAGERTABLE	EBCDIC ARRAY	01 DISPLAY	01 DISPLAY
MISCPARAM	HEX ARRAY	01 COMP-2	01 COMP

The ALGOL parameters all have a lower bound of 0, and no OCCURS are allowed at the 01 level for COBOL parameters.

TERMMSG contains a message delivered from or to be delivered to a terminal.

MCSMSG contains a message delivered from or to be delivered to the MCS.

PAGERTABLE, when dealing with a paged format, holds data for the library between initialization, input formatting and output formatting calls.

MISCPARAM contains eight six-character fields:

1. FUNCTION, digits 0-5, specifies which major function the library is to perform.
2. FORMAT CASE, digits 6-11, specifies which formatting code to use. This number is part of the information passed to the MCS during initialization.
3. TERMMSG SIZE, digits 12-17, specifies how many characters TERMMSG contains.
4. MCSMSG SIZE, digits 18-23, specifies how many characters MCSMSG contains.
5. ERROR NUMBER, digits 24-29, returns an error number to GEMCOS. When this field is greater than zero, the MCS will display an error message to the system monitor stations.
6. PRACTICE MODE, digits 30-35, when equal to 1 specifies that the station associated with this call is in practice mode.
7. PAGE FUNCTION, digits 36-41, requests an action of GEMCOS as result to a paged input or paged recall call.
8. STATION NUMBER, digits 42-47, the TCL number of the station that this message is coming from on input, or going to on output.

Details concerning the meanings and uses of each parameter are given below for each type of call. When the description of a library call does not mention a parameter, that parameter is not valid for the call and should not be used.

## Initialization

During initialization, GEMCOS calls the library to provide a list of format titles, case values and character requirements. This list is passed as a one dimensional EBCDIC array with one eighteen-character EBCDIC field and two six-character hex fields for each format. The first field contains the format title, which must match a title declared in the TCL, left justified with trailing blanks. The second and third fields contain the case value of the format and the number of characters required if this is an output or paged format.

An ALGOL library with one format named LIBFORMAT with a case index of 1 and that required 40 characters in the message it is formatting might declare its format list as:

```
VALUE ARRAY FORMATLIST
      ("LIBFORMAT      ",4"000001000040");
```

If a TP creates a message which contains fewer than the number of characters needed to be formatted correctly, the MCS appends blanks to the message before delivering it to the format library.

Input Parameters:

FUNCTION = 1

Output Parameters:

MCSMSG = The list of format titles, indices and character requirements

MCSMSG SIZE = The number of eight bit characters in MCSMSG

## Input Formatting

GEMCOS delivers a message from a station which the library modifies for delivery to a TP.

Input Parameters:

FUNCTION = 2

TERMMSG = The message received from a terminal

TERMMSG SIZE = The number of characters in TERMMSG

FORMAT CASE = The case index of the format to be applied to TERMMSG

PRACTICE MODE = 1 for a station in practice mode

STATION NUMBER = The TCL number of the station from which this message came



TERMMSG SIZE = The number of characters in TERMMSG

FORMAT CASE = The case index of the format to be applied to TERMMSG

PRACTICE MODE = 1 if this station is in practice mode

MCSMSG = The stored message for this dialogue

MCSMSG SIZE = The number of characters in MCSMSG

PAGERTABLE = The array which holds data pertaining to this dialogue

STATION NUMBER = The TCL number of the station involved in this paging dialogue

#### Output Parameters:

MCSMSG = The stored message, perhaps changed in pager input

PAGERTABLE = Updated data pertaining to this dialogue

ERROR NUMBER = 0 for no error  
> 0 when an error occurs

PAGEFUNC = 1 to cancel the dialogue  
2 to restart the dialogue with the original MCSMSG  
3 to format the requested page for delivery to a terminal  
4 to deliver the stored message to a TP  
5 to deliver a library output format to a terminal (similar to a forms request, but within PAGER)  
Any other value in PAGEFUNC will cancel the dialogue.

FORMAT CASE = The case index of an output format to send to the terminal when  
PAGEFUNC = 5

#### Output Parameters:

MCSMSG = The formatted message to be delivered to a TP

MCSMSG SIZE = The number of characters in MCSMSG

ERROR NUMBER = 0 for no error  
> 0 when an error occurs

### Output Formatting

GEMCOS delivers a message from a TP which the library modifies as necessary for delivery to a terminal.

#### Input Parameters:

FUNCTION = 3

MCSMSG = The message as built by a TP

MCSMSG SIZE = The number of characters in MCSMSG

FORMAT CASE = The case index of the format to be applied to MCSMSG

PRACTICE MODE = 1 for a station in practice mode

STATION NUMBER = The TCL number of the station to which this message is going

#### Output Parameters:

TERMMSG = Formatted message to be delivered to a terminal

TERMMSG SIZE = The number of characters in TERMMSG

ERROR NUMBER = 0 for no error  
> 0 when an error occurs

### Paged Format Initialization

GEMCOS delivers a message which generally contains data for multiple interactions with a terminal from a TP to the library. This message is stored over the course of these interactions, or dialogue, and may be read or changed during any format call during this dialogue.

A second array is available for storing any applicable dialogue information. This array, PAGERTABLE, can be used for passing data among the three paged formatting procedures, and is retained throughout a dialogue.

The initialization function is called first to allow the library to initialize PAGERTABLE in preparation for sending out the first page of the dialogue by the PAGED OUTPUT function. The initialization function may, but in most cases will not, modify MCSMSG.

#### Input Parameters:

FUNCTION = 4

MCSMSG = The entire message from a TP (stored message)

MCSMSG SIZE = The number of characters in MCSMSG

FORMAT CASE = The case index of the format to be applied to MCSMSG

PRACTICE MODE = 1 for a station in practice mode

STATION NUMBER = The TCL number of the station involved in this paging dialogue

#### Output Parameters:

MSCMSG = The stored message as modified by initialization

PAGERTABLE = Library defined data pertaining to this dialogue

ERROR NUMBER = 0 for no error  
> 0 when an error occurs

### Paged Input

Paged input's functions are more complex than those of the other paging procedures. GEMCOS delivers the message from a terminal, TERMMSG, and the stored message, MCSMSG, to the library. The library must then determine from TERMMSG which path the dialogue should follow next and inform GEMCOS via the PAGEFUNC field of MISCPARAM. The most likely result of a paged input call is for a new screen to be delivered to a terminal. In this case, paged input must determine which page is next, perhaps by looking for "+", "-", or other characters in TERMMSG as regular paging would do, or by accepting page requests by page number. Whichever page is selected, paged input must communicate the page number to paged output via PAGERTABLE.

Other possible results of the paged input call are cancellation of the dialogue, restarting the dialogue with the original stored message, transmitting the stored message to a TP or requesting that a library output format be delivered to the terminal without interrupting the dialogue. The five different results of a paged input call are determined by a number, one through five, placed in PAGEFUNC.

If the dialogue is not to be cancelled or restarted, data may be extracted from TERMMSG and used to update MCSMSG.

#### Input Parameters:

FUNCTION = 5

TERMMSG = The message received from a terminal

## Paged Output

GEMCOS delivers MCSMSG and PAGERTABLE to paged output which must determine which part of MCSMSG to format for delivery to the terminal and do so.

### Input Parameters:

FUNCTION = 6

MCSMSG = The stored message for this dialogue

MCSMSG SIZE = The number of characters in MCSMSG

FORMAT CASE = The case index of the format to be applied to MCSMSG

PRACTICE MODE = 1 for a station in practice mode

STATION NUMBER = The TCL number of the station involved in this paging dialogue

### Output Parameters:

TERMMSG = The message to be delivered to a terminal

TERMMSG SIZE = The number of characters in TERMMSG

ERROR NUMBER = 0 for no error  
> 0 when an error occurs

## Message Recall

Messages which do not use paged formats can be recalled and formatted by the FORMAT LIBRARY via regular output formatting calls. However, because the conventions of a paging dialogue are library defined, the library must provide special code if messages which use paged formats are to be recalled and formatted.

For a message requiring a paged format, the MCS first calls page initialization, function = 4, just as at the start of a dialogue. Next, the MCS calls on paged recall, function = 7, to format the first page of what would be dialogue with a terminal. GEMCOS will continue calling on paged recall and collecting the resulting formatted pages until a page is returned and PAGEFUNC = 6, denoting the last page. On each pass, paged recall must update TERMMSG, PAGERTABLE, and MISCPARAM so that the next page will be formatted on the next call. The details of deciding which page to format and how to determine when the last page has been formatted are left to the library developer.

### Input Parameters:

FUNCTION = 7

MCSMSG = The entire message from a TP

MCSMSG FORMAT = The number of characters in MCSMSG

FORMAT CASE = The case index of the format to be applied to MCSMSG

PRACTICE MODE = 1 for a station in practice mode

PAGERTABLE = Data pertaining to the recalled message formatting

STATION NUMBER = The TCL number of the station involved in this paging dialogue

#### Output Parameters:

TERMMSG = One page of the formatted message

TERMMSG SIZE = The number of characters in TERMMSG

PAGERTABLE = Data pertaining to this recall dialogue

ERROR NUMBER = 0 for no error  
> 0 when an error occurs

PAGEFUNC = 6 when TERMMSG contains the last page of the recall dialogue

## Format Library Updating

The only way to update a format in a library is to delink the library from GEMCOS and then relink to a new library containing the revised format. At present, unfortunately, there is no way to delink a library from GEMCOS short of bringing the MCS down. This will be remedied in a future release.

## Notes

Additional information concerning libraries may be found in the B 6000 Mark 3.1 P & D Notes, form number 5011257.

## DEMONSTRATION FILES

GEMCOS/FORMATLIB/TCL contains a complete description of the stations, TP, and formats necessary to run the format library demonstration.

GEMCOS/FORMATLIB/PATCH contains all references to stations from GEMCOS/FORMATLIB/TCL to simplify modifying the network for a particular site.

GEMCOS/SAMPLE/FORMATTP is the object code for the simple transaction processor used in the format library demonstration.

GEMCOS/FORMATTP/SYMBOL is the ALGOL source code for the GEMCOS/SAMPLE/FORMATTP.

GEMCOS/SAMPLE/ALGOLLIB and GEMCOS/SAMPLE/COBOLLIB are two versions of the source for compilation of the format library itself. Both provide functionally similar formats to the MCS and either can be compiled as GEMCOS/FORMATLIBRARY to run the demonstration.

These files also serve as examples of the structure necessary to write a format library and can be easily modified to match a site's requirements.

GEMCOS/FORMATLIBRARY is the object file compiled from one of the sample format libraries and provides the formatting procedures to the MCS. GEMCOS/EDITOR, the standard editor, is used in the demonstration.

Generally, only the station names in GEMCOS/FORMATLIB/PATCH need to be changed to fit the demonstration TCL to a particular site. Compile this patch file and GEMCOS/FORMATLIB/TCL together using the GEMCOS/UTILITY and run the MCS with the resulting DATA-COM/QUEUE files.

After switching a control station to GEMCOS, enable all and then enter the forms request DEMO. GEMCOS will return the first of the library formatted screens and the demonstration is self-explanatory from that point.

Note that the TP, TCL, and format libraries included on the release tape for using format libraries serve only as examples and are not supported by Burroughs.



## **SECTION 7**

# **ACCESS CONTROL**

Access control is optionally available on a station-by-station basis and is declared in the Transaction Control Language (TCL).

The user can declare in the <system section> a list of valid access keys. Each access key denotes a different class of user of that system. The user associates each access key with a list of message keys which that class of user is allowed to enter into the system.

Each station in the network may be declared with either SIGNON = TRUE or SIGNON = FALSE. Stations specifying SIGNON = TRUE may declare a list of valid access keys which defines the classes of users which may use that station. (If the list of valid access keys is omitted, all classes of users may use the station.) Operators at a SIGNON = TRUE station must identify themselves to the Message Control System (MCS) as belonging to one of the valid classes of users declared for that station.

If SIGNON = FALSE is specified for a station, no special identifying procedure need be performed by operators of the station.

## **INTERNAL MCS ACCESS CONTROL MECHANISM**

The user may wish to have the MCS process all access control transactions. This is done by defining special LOGON and LOGOFF message keys in the GLOBAL section of the TCL. A terminal operator may then identify himself or herself to the MCS by inputting a transaction with the special LOGON message key whose data is one of the valid access keys defined for that station. If the MCS determines that the access key is indeed valid for that station, it allows the log-on, and the terminal operator may input any transaction valid for that class of user. If the operator enters an invalid access key, the MCS responds by sending an error message to that effect so that a second attempt can be made. If there are two successive failures at logging onto a station, the station is disabled, and an error message is sent to all System Monitor stations. The station must then be enabled at a System Network Control station before it can be used again.

## **USER-SUPPLIED ACCESS CONTROL MODULE**

The user has the option of declaring his or her own Access Control module to handle log-on and log-off transactions. It is declared the same as any other Transaction Processor (TP) in the TCL, with the following additional syntax:

**ACCESSCONTROLPROGRAM = TRUE.**



All message keys defined for this program are assumed by the MCS to be log-on or log-off message keys. The user-supplied Access Control module is identical to any other User program in its interface to the MCS, and as long as it follows the interface conventions, it can do any type of checking desired. Users might determine that this program should not be defined as PERMANENT = TRUE in the TCL. In this way, access control would only be active when a log-on or log-off message is received from a station. If the Access Control module determines that the attempted log-on or log-off is invalid, it may inform the station. If the attempt is a valid one, the program must inform the MCS with special log-on and log-off passes of control. The Access Control module is not restricted to logging on or logging off the originating station; it may request that the MCS validate the log-on or log-off of any station in the system. In addition, the Access Control module may request that all stations in a "system" be logged off.

The log-on pass of control consists of:

1. Value 14 in MSG-ACTION (WORD[1] of COMMON).
2. A valid access key in the text area of COMMON.
3. The length of the access key in MSG-LENGTHIN (WORD[7] of COMMON).
4. The station number of the station to be logged on in MSG-DEST (WORD[4] of COMMON). (0 means log-on station that originated the message).

The log-off pass of control consists of:

1. Value 16 in MSG-ACTION (WORD[1] of COMMON).
2. The station number of the station to be logged off in MSG-DEST (WORD[4] of COMMON). (0 means log-off station that originated the message).

The "system" log-off pass of control consists of value 21 placed in MSG-ACTION (WORD[1] of COMMON). This logs off all stations in this system.

The MCS performs the same checks on an access key passed to it by an Access Control module that would be performed under the internal MCS access control mechanism.

If the MCS determines that the log-on or log-off pass of control is valid, it returns a NORMAL response to the Access Control module (MSG-ACTION (COMMON[1]) = 6, MSG-LENGTHIN (COMMON[7]) = 0). If the log-on or log-off pass of control is invalid, the MCS returns a security error to the Access Control module (MSG-ACTION (COMMON[1]) = 5, MSG-LENGTHIN (COMMON[7]) = error code). It is up to the Access Control module to inform the terminal operator of the result of the attempted log-on or log-off. The possible error codes are as follows.

- 143 - Undefined ACCESSKEY
- 144 - ACCESSKEY not allowed at station
- 145 - Too many users
- 146 - User already logged on
- 147 - Another user already logged on
- 148 - Station is assigned

If a non Access Control Program attempts to pass one of the special log-on or log-off requests to the MCS, it will receive a special security error (MSG-ACTION (COMMON[1]) = 5, MSG-LENGTHIN (COMMON[7]) = 133).

## Station Bits

The user-supplied Access Control module has the added capability of changing the station bits of the station to which the user is trying to log on or log off. The Access Control module may change the station bits in COMMON[8] prior to passing control to the MCS for log-on or log-off verification. If the log-on or log-off is valid, the MCS moves whatever is in the station bits in COMMON into the station bits for the station that is attempting the log-on or log-off. This means that users can set up their own conventions, utilizing constant station bits (declared in the TCL) in conjunction with variable station bits (set at log-on and log-off time). If the Access Control module is a Process program, it should use WORD[12] of the CONTROLWORDS array for the same purpose. In the case of the "system" log-off pass of control, the contents of the station bits word are used as a mask applied to the station bits of all stations in the system.

## Individual Identification

If INDIVIDUALID is specified TRUE for a station in the TCL, the MCS accepts a 2-word identifier from the Access Control module with each valid log-on request from that station. The identifier should be placed into MSG-USERID1 (COMMON[21]) and MSG-USERID2 (COMMON[22]) by the Access Control module before requesting log-on verification by the MCS. If the log-on is valid, the contents of MSG-USERID1 (COMMON[21]) and MSG-USERID2 (COMMON[22]) are saved by the MCS and returned to a TP each time the user inputs a transaction at that station. The individual-ID is removed when the user logs off the station. If the Access Control module is a Process program, it should use WORD[25] and WORD[26] of the CONTROLWORDS array for the same purpose.



## SECTION 8

# NETWORK MANAGEMENT AND CONTROL

Network management and control functions are handled by the Control module. This module is invoked from the Process-everything module. It may be invoked once and remain in existence for the entire time the Message Control System (MCS) is running, or it may be invoked each time it is needed. This depends on whether it was specified in the TCL that `CONTROLPERMANENT = TRUE`.

The Control module has two parameters, both events. One of these events is used by Process-everything to wake up Control; the other is used by Control to wake up Process-everything. When Process-everything detects a message in the Control queue, it causes a Control event. Control then wakes up and retrieves the message (using the Global procedure `GETMESSAGE`). After processing the message, Control responds, if appropriate (using the Global procedure `SENDMESSAGE`), causes the Process-everything `MASTEREVENT`, and either goes to End-of-Job or goes to sleep, depending upon the value of `CONTROLPERMANENT`. The Control module, then, operates exactly as if it were a Process program, and in fact, the MCS treats it as if it were.

In order to effect certain levels of control over the data communications processing environment, each institution or application (hereafter referred to as "system") has the capability to identify certain stations as System Network Controllers or System Monitors. A System Network Controller (SPO) is defined as a station that can issue System Control messages and System Definition messages. A System Monitor is defined as a station that receives messages of a system nature. A station can be defined as both System Network Controller and System Monitor, or it can be individually assigned as either a System Network Controller or a System Monitor. At least one System Network Controller and one System Monitor must be defined.

System Control messages make it possible for Network Control stations to affect and/or monitor system operations. After the MCS is initialized, the only stations enabled for input and output are Network Control stations. It is through these stations that System Control messages are sent to initiate the system, or parts of the system. It is also possible to enter System Control messages (? messages) at the computer's supervisory console station using the SM (Send Message) command. The syntax for entering such a command is as follows:

```
<mix no>SM:? <text>
```

The `<mix no>` is the job number of the MCS. When this command is accepted by the MCS, the first part of any response is displayed at the console. System Control messages may also be issued by an Access Control Program using the Transaction Processor TP-to-TP routing mechanism. The Format Generator is capable of issuing `UPDATE FORMAT` and `STATUS FORMAT` commands. No other programs have this capability.

Some Control messages are for informational purposes only; that is, they request information which is contained in the MCS tables. Other messages affect or change the contents of those tables.

The syntax of System Control messages is contained in the remainder of this section.

#### NOTE

Beginning with "System Defined Input Message," the material contained in this section is presented in the following order: 1) Message, 2) Syntax, and 3) Semantics.

## SYSTEM DEFINED INPUT MESSAGE

Syntax:

```
<system defined input message> ::=  
    <assign input message> /  
    <change input message> /  
    <close input message> /  
    <compute input message> /  
    <dollar-sign input message> /  
    <intercept input message> /  
    <mode input message> /  
    <quit input message> /  
    <refresh output message> /  
    <switch input message>
```

Semantics:

<System defined input message>s allow a station to affect its message handling, monitor other stations, indicate operability of its station, control the data communication system, and perform arithmetic calculations.

## ASSIGN INPUT Message

Syntax:

```
<assign input message> ::=  
    ASSIGN <optional station specification> <comment> TO  
    <program name>  
  
<optional station specification> ::=  
    <station specification> / <empty>
```

### Semantics:

The <assign input message> is used to attach any station to a Transaction Processing program. The user must be logged onto the station before the assignment is allowed. If the user does not have access to any one of the message keys associated with the Transaction Processing program, the assignment is not allowed, nor is it allowed on a multilog-on station. After the station is successfully assigned, all input messages except <system defined input message>s and <system control message>s are sent to the Transaction Processing program without the need of the normal message key. If such an input message is headed by a normal message key, however, the appropriate module and function indices and item count are passed to the TP; and, if an editor exists for the Transaction Processing program, input formatting is also performed if required.

If the assign message is valid, a copy of it is sent to the Transaction Processing program.

A station is "unassigned" by means of the <close input message>. If the <optional station specification> is not <empty> then the station specified is assigned to the program rather than the station entering the assign message.

The responses to an <assign input message> are:

1. If assign message valid:

STATION <station name> (<station number>) IS  
ASSIGNED TO <program name>  
IN <mode> MODE

2. If assign message invalid:

ASSIGN MESSAGE INVALID  
PLEASE RE-ENTER

3. If station already assigned:

THIS STATION HAS NOT BEEN CLOSED

## CHANGE INPUT Message

### Syntax:

<change input message> ::=  
CHANGE <comment> <to> <mode> <comment>

<mode> ::= <data mode> / <practice mode>

<data mode> ::= DATA

<practice mode> ::= PRACTICE

#### Semantics:

The <change input message> is used to change the mode that a station is in. Normally a station is in <data mode>, which means the input messages from this station are not of a practice or test nature (indicated by <practice mode>). The station's mode is passed to a Transaction Processing program via the COMMONAREA for use as required.

The response to the <change input message> is:

1. If change message valid:

STATION <station name> (<station number>) IS  
IN <mode> MODE

2. If change message invalid:

PLEASE REPEAT CHANGE MESSAGE

### CLOSE INPUT Message

#### Syntax:

<close input message> ::=  
CLOSE <optional station specification> <comment>

#### Semantics:

The <close input message> is used to unassign an assigned station. If the station is assigned to a program, a copy of the close message is sent to the Transaction Processing program.

A user may not log off an assigned station until the station is closed.

If <optional station specification> is not <empty> then the station specified is closed rather than the station entering the assign message.

The response to the <close input message> is:

STATION <station name> (<station number>) HAS BEEN CLOSED  
STATION IS IN <mode> MODE

### COMPUTE INPUT Message

#### Syntax:

<compute input message> ::=  
COMPUTE <output format> <arithmetic expression>  
<compute comment>

```

<output format> ::=
    <decimal output> /
    <binary output> / <octal output> /
    <quaternary output> /
    <hexadecimal output> /
    <BCL output> /
    <ASCII output> /
    <EBCDIC output>

<decimal output> ::= DECIMAL / DEC / <empty>

<binary output> ::= BINARY / BIN

<octal output> ::= OCTAL / OCT

<quaternary output> ::= QUATERNARY / QUA

<hexadecimal output> ::= HEXADECIMAL / HEX

<BCL output> ::= BCL

<ASCII output> ::= ASCII / ASC

<EBCDIC output> ::= EBCDIC / EBC

<arithmetic expression> ::=
    <term> / <adding operator> <term> /
    <arithmetic expression> <adding operator> <term>

<term> ::= <factor> / <term> <multiplying operator>
    <factor>

<factor> ::=
    <primary> /
    <factor> <exponentiation operator> <primary>

<primary> ::= <unsigned number> / <string> / <operand>

<adding operator> ::= + / -

<multiplying operator> ::= * / <slash> / DIV / MOD

<exponentiation operator> ::= **

<operand> ::=
    <function expression> /
    (<arithmetic expression>) /
    <arithmetic constant identifier>

```



<function expression> ::=  
    <function name> (<arithmetic expression>)

<function name> ::=  
    INTEGERT /  
    INTEGER /  
    ENTIER /  
    ABS /  
    NABS /  
    SIGN /  
    SIN /  
    COS /  
    TAN /  
    COTAN /  
    ARCSIN /  
    ARCCOS /  
    ARCTAN /  
    SINH /  
    COSH /  
    TANH /  
    LOG /  
    LN /  
    EXP /  
    ERF /  
    GAMMA /  
    LNGAMMA /  
    SQRT

<arithmetic constant identifier> ::= PI / E / RANDOM

PI ::= 3.1415926535897932384626

E ::= 2.7182818284590452353603

RANDOM ::=  
    [a member of a set, 0.0 LEQ member LEQ 1.0, whose  
    members have equal probability of occurring.]

<compute comment> ::= ; <comment>

<comment> ::=  
    [any sequence of <letter>s, <digit>s or <single  
    space>s with the exception of the word "to" in  
    certain syntax definitions] / <empty>

### Semantics:

The <compute input message> allows arithmetic calculations to be performed at a station. Station accessibility is controlled by the <master compute statement> in the <system section> or the <compute statement> in the <station section>.

Function names are defined as follows (where <ae> indicates an arithmetic expression):

Name	Definition
INTEGERT	Integerize <ae> by truncation
INTEGER	Entier (<ae> + 0.5)
ENTIER	Largest integer not greater than <ae>
ABS	Absolute value of <ae>
NABS	Negative absolute value of <ae>
SIGN	+1 If <ae> GRT 0, 0 if <ae> EQL 0, -1 if <ae> LSS 0
SIN	Sine of <ae>, <ae> in radians
COS	Cosine of <ae>, <ae> in radians
TAN	Tangent of <ae>, <ae> in radians
COTAN	Cotangent of <ae>, <ae> in radians
ARCSIN	Principal value of arc sine of <ae>, -1 LSS <ae> LSS 1
ARCCOS	Principal value of arc cosine of <ae>, -1 LSS
ARCTAN	Principal value of arc tangent of <ae>, <ae> in radians
SINH	Hyperbolic sine of <ae>, <ae> LEQ 68059.0
COSH	Hyperbolic cosine of <ae>, <ae> LEQ 68059.0
TANH	Hyperbolic tangent of <ae>
LOG	Logarithm to base 10 of <ae>, <ae> GTR 0.0
LN	Logarithm to base e of <ae>, <ae> GTR 0.0
EXP	Base e raised to the <ae> power, <ae> LEQ 68059.0
ERF	Standard error function of <ae>
GAMMA	Gamma function of <ae>, 0.0 LSS <ae> LEQ 8461.0
LNGAMMA	Natural logarithm of the gamma function at <ae>, 0.0 LSS <ae> LEQ 8461.0.
SQRT	Square root of <ae>, <ae> GEQ 0.0

The sequence in which operations are performed is determined by the precedence of the operators involved. The order of precedence is:

1. FIRST: \*\*
2. SECOND: \* / MOD DIV
3. THIRD: + -

When operators have the same order of precedence, the sequence of operation is determined by order of their appearance, from left to right. Parentheses can be used in normal mathematical fashion to override the usual order of precedence. The operators are defined as follows:

1. <Adding operator>s are defined to be:

- a. + : REAL ADDITION
- b. - : REAL SUBTRACTION

2. <Multiplying operator>s are defined to be:

- a. \* : REAL MULTIPLICATION
- b. / : REAL DIVISION
- c. DIV INTEGER DIVISION:  $Y \text{ DIV } Z = \text{SIGN}(Y/Z) * \text{ENTIER}(\text{ABS}(Y/Z))$
- d. MOD REAL REMAINDER DIVIDE:  $Y \text{ MOD } Z = Y - (Z * (\text{SIGN}(Y/Z) * \text{ENTIER}(\text{ABS}(Y/Z))))$

3. <Exponentiation operator> is defined to be:

\*\* REAL EXPONENTIATION,  $10 ** 2 = 100$

4. <Unsigned number> is defined to be:

- a. largest number that is acceptable to the integer function:  $30223145490365729367543 = 8 ** 26 - 1$
- b. smallest real number:  $1.93854585713758583355640 \text{ E-}29581 = 8**-32755$
- c. largest real number:  $1.94882838205028079124469 \text{ E } 29603 = (1 - 8 ** -26) * 8 ** 32780$

5. The response to the <compute input message> is:

COMPUTE ANSWER IS <number>

The <number> can be an integer, a real number, or an engineering notation (e.g.  $1.3998@ 05 = 1.3998 * 10 ** 5 / 1.3998@ - 00005 = 113998 * 10 ** -5$ ).

Example 1:

```
COMPUTE 10 * 10
COMPUTE ANSWER IS 100
```

Example 2:

```
COMPUTE 180 / PI; DEGREES PER RADIANT
COMPUTE ANSWER IS 57.2957795130823208768
```

Example 3:

```
COMPUTE SQRT (4) ** 2 + 4
COMPUTE ANSWER IS 8
```

## \$ INPUT Message

Syntax:

```
<dollar sign input message> ::=  
    $. <service digit>.<optional comment> /  
    $. <service digit> <station specification>  
    <optional comment>
```

```
<service digit> ::= 1 / 2
```

Semantics:

The <dollar sign input message> allows a station to change its operating status with the MCS. It also allows one station to put another out of service and enables SPECIAL = TRUE stations to put other stations back in service.

A <service digit> of 1 indicates the station wishes to go OUT-OF-SERVICE. At that time, the station is marked as inoperative, and the message, including any comment, is routed to all Monitor stations. Output for this station is subsequently queued, pending resumption of service, or for a period of 15 minutes, whichever is sooner. Output to alternate stations is not attempted during this type of OUT-OF-SERVICE status.

A <service digit> of 2 indicates the station wishes to resume its former IN-SERVICE status. This action can only be performed by stations described with SPECIAL = TRUE in the <station section> of the TCL. OUT-OF-SERVICE stations with SPECIAL = FALSE expressed or implied can only resume their former IN-SERVICE status via a System Control message issued from a Network Control station or expiration of the 15-minute time period.

Example:

```
$1. I AM CHANGING FORMS.
```

```
$2. PUT ME BACK IN SERVICE NOW.
```

```
$1 S STATION1. GOING OUT OF SERVICE
```

## INTERCEPT INPUT Message

Syntax:

```
<intercept input message> ::=  
    INTERCEPT <cancel> STATION <station ident>  
    <intercept myuse> <comment>
```

```
<cancel> ::= - / MINUS / <empty>
```

```
<station ident> ::= <station name> / (<station number>)
```

```
<intercept myuse> ::= IN / OUT / IO / <empty>
```

#### Semantics:

The <intercept input message> is used by an INTERCEPT = TRUE station. This message is used to start or stop interception of another station's input and/or output messages. The <intercept myuse> part specifies which messages are to be intercepted. <Intercept myuse> of <empty> assumes interception of output messages.

A single station may intercept many stations but may be intercepted by only one station.

The responses to an <intercept input message> are:

1. If intercept message valid and <cancel> is <empty>:

INTERCEPT OF STATION <station name> (<station number>) STARTED 11/12/  
73 1345

2. If intercept message valid and <cancel> is - or minus:

INTERCEPT OF STATION <station name> (<station number>) STOPPED 01/03/  
74 1128

3. If intercept message invalid, and format is bad:

INTERCEPT MESSAGE INVALID

4. If intercept message invalid, and station has already been intercepted:

INTERCEPT MESSAGE INVALID STATION ALREADY BEING INTERCEPTED BY  
STATION <station name> (<station number>)

## MODE INPUT Message

#### Syntax:

<mode input message> ::= MODE <comment>

#### Semantics:

The <mode input message> is used to determine the mode my station is in.

The response to a <mode input message> is:

1. If station assigned:

STATION <station name>(<station number>) IS ASSIGNED TO <program name>  
IN <mode> MODE

2. If station not assigned:

STATION <station name> (<station number>) IS IN <mode> MODE

## QUIT INPUT Message

Syntax:

<quit input message> ::= QUIT <comment>

Semantics:

The <quit input message> is used to disconnect a dial-in station from the Data Communication System. If the station is assigned to a Transaction Processing program, a copy of the quit message is forwarded to that program.

## REFRESH OUTPUT Message

Syntax:

<refresh output message> ::=  
REFRESH <optional disk address> <optional destination>

<optional disk address> ::= <integer> / <empty>

<optional destination> ::=  
TO <station specification> / <empty>

Semantics:

The <refresh output message> can be used to recall the last output audited for the terminal. Note that this is not necessarily the last output actually received at the terminal. If an integer follows the REFRESH command as data, it is assumed that this is the disk address of the base record of an output message, and the specified message is recalled. This is the method by which possible duplicate outputs may be recalled by the user after recovery. A recall TP need not be declared in the TCL in order to use the REFRESH command.

If a destination follows the REFRESH command as data, this implies that the destination terminal is to be refreshed. If an integer is used as a disk address in this case, it is assumed that this is the disk address of the base record of an output message from the destination terminal, and the specified message is recalled at the destination terminal.

REFRESH can only result in the requeuing of a message to its original destination station. If a disk address causes retrieval of a message not originally destined to the station for which the request was intended, the request is invalid and is not honored.

Example:

```
REFRESH 123
REFRESH
REFRESH TO S TTY2
REFRESH 97 TO S(3)
```

## SWITCH INPUT Message

Syntax:

```
<switch input message> ::=  
    SWITCH <station specification> TO <system  
    specification> / SWITCH TO <system specification>
```

Semantics:

The <switch input message> allows the operator to switch a station to a new system. If no station is specified, the operator's station is switched. Only a Network Control Station may specify that a station other than itself be switched. The system to which station currently belongs is backed up in the Control file so that it remains across EOJs until a new generation.

Example:

```
ENTER: SWITCH TO SYSTEM (2)  
RESPONSE: SYSTEM SWITCH COMPLETED
```

POSSIBLE ERRORS:

```
SWITCH CANNOT BE DONE: STATION ASSIGNED  
SWITCH CANNOT BE DONE: STATION LOGGED ON  
SWITCH CANNOT BE DONE: STATION BEING INTERCEPTED  
SWITCH CANNOT BE DONE: STATION NOT DECLARED IN SYSTEM  
SWITCH CANNOT BE DONE: SWITCHING STA INTERCEPTING  
SWITCH CANNOT BE DONE: SWITCHING STA MUST BE SPO
```

## SYSTEMCONTROLMESSAGES

Syntax:

```
<system control message> ::=  
    <QM> <control request list>
```

```
<QM> ::= ?
```

```
<control request list> ::=  
    <control request> /  
    <control request list> ; <control request>
```

```
<control request> ::=  
    <add request> /  
    <attach request> /  
    <change request> /  
    <clear request> /  
    <disable request> /  
    <dump request> /
```

```

<enable request> /
<interrogate request> /
<move request> /
<recover request> /
<release request> /
<run program request> /
<status request> /
<subtract request> /
<system request> /
<table request> /
<update request> /
<where request> /
<zip request>

```

#### Semantics:

System Control messages make it possible for Network Control stations to affect and/or monitor system operation. After the MCS is initialized, the only stations enabled for input and output are the Network Control stations. It is through these stations that System Control messages are sent to initiate the total network, individual systems, or parts of individual systems.

As in the case of System End-of-Job, a <system request> affects the total system. The other requests may affect groups of elements in the system or a particular element in the system. Requests that affect groups of elements contain an <inclusive identification list>, whereas a request that affects a particular element contains a <specific identification list>.

#### NOTE

In order to help the reader understand the syntax of the individual <system control message>, the definitions that comprise an <inclusive identification list> are defined below instead of as they occur in the syntax.

### Inclusive Identification List

#### Syntax:

```

<inclusive identification list> ::=
    <inclusive identification> /
    <inclusive identification list> ,
    <inclusive identification>

```

```

<inclusive identification> ::=
    STATIONS / SS
    PROGRAMS / PS
    INPUTQUEUES / IS
    COMMONS / CS
    LINES / LS
    TEST /
    ENABLED /

```



SYSTEMSPOS /  
SYSTEMMONITORS /  
SPECIAL /  
MCSS

## Specific Identification List

Syntax:

```
<specific identification list> ::=  
    <specific identification> /  
    <specific identification list> ,  
    <specific identification>
```

```
<specific identification> ::=  
    <program specification> /  
    <system specification> /  
    <input queue specification> /  
    <station specification> /  
    <area specification> /  
    <common specification> /  
    <line specification> /  
    <cluster specification> /  
    <DCP specification> /  
    <message key specification> /  
    <message id specification>
```

```
<program specification> ::=  
    PROGRAM <program designator> / P <program designator>
```

```
<program designator> ::=  
    <program name> <stack designator> /  
    (<program number>) <stack designator>
```

```
<stack designator> ::=  
    - <stack number> / <empty>
```

```
<stack number> ::= <integer>
```

```
<system specification> ::=  
    SYSTEM <system designator>
```

```
<system designator> ::=  
    <system name> / (<system number>)
```

```
<input queue specification> ::=  
    INPUTQUEUE  
    <input queue designator> / I <input queue designator>
```

```

<input queue designator> ::=
    <input queue name> / (<input queue number>)

<station specification> ::=
    STATION <station designator> /
    S <station designator>

<station designator> ::=
    <station name> / (<station number>)

<area specification> ::=
    AREA <area designator> / A <area designator>

<area designator> ::=
    <area name> / (<area number>)

<common specification> ::=
    COMMON (<integer>) / C (<integer>)

<line specification> ::=
    LINE <line designator> / L <line designator>

<line designator> ::=
    (<DCP address>) / <station name>

<DCP address> ::=
    <DCP>:<cluster>:<adapter>

<cluster specification> ::=
    CLUSTER (<cluster address>)

<cluster address> ::= <DCP>:<cluster>

<DCP specification> ::= DCP (<DCP>)

<DCP> ::= <integer>

<cluster> ::= <integer>

<adapter> ::= <integer>

<message key specification> ::=
    MKE <message key>

<message id specification> ::=
    MID <message identifier>

```

## ADD Request

### Syntax:

```
<add request> ::=  
    <ADD <station specification> TO  
    <line specification> <optional terminal name change>  
  
<optional terminal name change> ::=  
    AS <string> / <empty>
```

### Semantics:

The ADD command allows a station to be added to a line. The ADD of a station with an existing line assignment will function as a MOVE network control command. The station must first be disabled. The <optional terminal name change> allows the station to be associated with a new terminal. This new terminal must be a valid NDL terminal name.

## ATTACH Request

### Syntax:

```
<attach request> ::=  
    ATTACH <attach specification list>  
  
<attach specification list> ::=  
    <attach specification> /  
    <attach specification list>, <attach specification>  
  
<attach specification> ::=  
    <station specification> / <line specification>  
    <DCP specification>
```

### Semantics:

The ATTACH command is used to allow GEMCOS to gain control of stations which are not currently attached. This can happen after a datacom reconfiguration or a DCP beginning or end of job while GEMCOS is running. When the <attach specification> is a <line specification> all stations belonging to GEMCOS on that line will be attached. When the <attach specification> is a <DCP specification>, any stations which were last attached to GEMCOS through that DCP and are now not attached will be attached.

### Example 1:

```
MESSAGE:  ?ATTACH S(1), L TD2  
RESPONSE: #1430 STATION (1) ATTACHED  
          #1430 STATION (2) ATTACHED
```

Example 2:

```
MESSAGE:  ?ATTACH DCP (0)
RESPONSE: #1432 STATION (1) ATTACHED
          #1432 STATION (2) ATTACHED
```

## CHANGE Request

Syntax:

```
<change request> ::=
    CHANGE <change specification list>

<change specification list> ::=
    <change specification> /
    <change specification list>, <change specification>

<change specification> ::=
    <station change> /
    <input queue change> /
    <program change> /
    <system change> /
    <line change> /
    <cluster change>

<station change> ::=
    <station specification>
    <station change object>

<station change object> ::=
    <object> /
    <alternate object> /
    <address object> /
    <line delete object> /
    <backspace object> /
    <end of message object> /
    <control char object> /
    <output object> /
    <station host name object> /
    <station your name object> /
    <top object> /
    <que object> /
    <monitor object> /
    <format monitor object> /
    <my use object> /
    <logical ack object> /
    <select object>
```

```

<bot object> ::=
    bot <to> NEXT /
    BOT <to> BOTBOT /
    BOT <to> TOP /
    BOT <to> <integer>

<to> ::= TO / =

<alternate object> ::=
    ALTERNATE <to> <station designator> /
    ALTERNATE <to> - <station designator>

<address object> ::=
    ADDRESS <to> <address> /
    ADDRESS <to> (<address>, <address>)

<address> ::=
    <EBCDIC string> / <hex string>
    [maximum length of 3 bytes]

<line delete object> ::=
    LINEDELETE <to> <1-byte string>

<backspace object> ::=
    BACKSPACE <to> <1-byte string>

<end of message object> ::=
    ENDOFMESSAGE <to> <1-byte string>

<control char object> ::=
    CONTROLCHAR <to> <1-byte string>

<output object> ::=
    OUTPUT <to> DIRECT /
    OUTPUT <to> NORMAL /
    OUTPUT <to> CLOSE

<top object> ::= TOP <to> <integer>

<queue object> ::= QUE <to> <integer>

<monitor object> ::=
    MONITOR <to> <logical value>

<format monitor object> ::=
    FMTMONITOR <to> <logical value>

```

```

<my use object> ::=
    MYUSE <to> 1 /
    MYUSE <to> 2 /
    MYUSE <to> 3

<logical ack object> ::=
    LOGICALACK <to> <logical value>

<select object> ::=
    SELECT START <to> <1-byte string>
    STOP <to> <1-byte string>

<station host name object> ::=
    STATIONHOSTNAME <to> <identifier>

<station your name object> ::=
    STATIONYOURNAME <to> <identifier>

<input queue change> ::=
    <input queue specification >
    <input queue change object>

<input queue change object> ::=
    <BOT object> / <time limit object> /
    <queue depth object> / <queue object> /
    <rerun object>

<time limit object> ::=
    TIMELIMIT <to> <integer>

<queue depth object> ::=
    QUEUEDEPTH <to> <integer>

<rerun object> ::=
    RERUN <to> <logical value>

<program change> ::=
    <program specification> <program change object>

<program change object> ::=
    <multiple inputs object> /
    <permanent object> /
    <title object> /
    <control bit object> /
    <max copies object> /
    <min copies object> /
    <host name object> /
    <subsystem name object> /
    <subspaces value object> /
    <declared priority value object>

```

```

<multiple inputs object> ::=
    REN <to> <logical value>

<permanent object> ::=
    PER <to> <logical value>

<title object> ::=
    TITLE <to> <title specification>

<title specification> ::=
    <string > / <optional user code password>
    <generalized identifier> <pack specification>

<control bit object> ::= BIT <to> <integer>

<max copies object> ::=
    MAXCOPIES <to> <integer>
    [number must be LEQ max copies value defined in TCL]

<min copies object> ::=
    MINCOPIES <to> <integer>

<host name object> ::=
    HOST TO <identifier>

<subsystem name object> ::=
    SUBSYSTEM TO <identifier>

<subspaces value object> ::=
    SUBSPACES TO <integer>

<declared priority value object> ::=
    DECLAREDPRIORITY TO <integer>

<system change> ::=
    <system specification> <system change object>

<system change object> ::=
    <host name> / <subsystem name>

<line change> ::=
    <line specification> <line change object>

```

```

<line change object> ::=
    <to> (<dcf address>)
    <type object> /
    <C2 object> /
    <BCBI object> /
    <SCSA object> /
    <CI object> /
    <IR object> /
    <MR object> /
    <OR object> /
    <MODEM object>

<type object> ::=
    TYPE <to> <integer> / TYPE <to> <string>

<C2 object> ::=
    C2 <to> <integer> / C2 <to> <string>

<BCBI object> ::=
    BCBI <to> <integer> / BCBI <to> <string>

<SCSA object> ::=
    SCSA <to> <integer> / SCSA <to> <string>

<CI object> ::=
    CI <to> <integer> / CI <to> <string>

<IR object> ::=
    IR <to> <integer> IR <to> <string>

<MR object> ::=
    MR <to> <integer> / MR <to> <string>

<OR object> ::=
    OR <to> <integer> / OR <to> <string>

<MODEM object> ::= MODEM <to> <string>

<cluster change> ::=
    <cluster specification> <cluster change object>

<cluster change object> ::=
    <to> (<cluster address>)

```



### Semantics:

**STATION CHANGES.** Station change requests allow changes to TCL, DCP, or run-time related attributes for the station. Alternate, address, my-use, logical-ack, stationhostname, and stationyourname attributes are described in the TCL documentation (refer to section 3). Various control characters for the station may be changed provided the NDL requests are written to handle the changes. Also, run-time station queue attributes and station monitor activities may be changed. Appendix D contains the format of the station monitor listing.

**INPUTQUEUE CHANGES.** The time limit and queue depth attributes as described in the TCL documentation may be changed with an Input queue change request. Additionally, run-time Input queue attributes such as the BOT pointer and Input queue rerun state may be changed.

**PROGRAM CHANGES.** All program changes relate to TCL attributes for the program. A description of these attributes can be found in the TCL documentation (refer to section 3).

**LINE AND CLUSTER CHANGES.** Line and cluster changes allow for changes to the network configuration and changes to the DCP-related line attributes.

The Select Start and Stop refer to starting and stopping characters in the Data Line Monitor output for a line analyzer station. The <output object> directs line analyzer output either to printer backup (NORMAL) or to a line printer (DIRECT).

#### Example 1:

```
MESSAGE:  ?CHANGE S TTY2 BOT TO
          TOP
RESPONSE:  # STATION TTY2 BOT
          CHANGED
          FROM 86 TO 97
```

#### Example 2:

```
MESSAGE:  ?CHANGE I(13) QUEUEDEPTH
          TO 5
RESPONSE:  # INPUTQUEUE (13)
          QUEUEDEPTH CHANGED
          FROM 3 TO 5
```

#### Example 3:

```
MESSAGE:  ?CHANGE P(2) TITLE TO
          "PROG1"
RESPONSE:  # PROGRAM (2) TITLE
          CHANGED
          FROM TEST/PROG. TO
          PROG1.
```

## CLEAR Request

Syntax:

```
<clear request> ::=
    CLEAR <clear identification list>

<clear identification list> ::=
    <clear identification> /
    <clear identification list>, <clear identification>

<clear identification> ::=
    <station specification> /
    <program specification> / <line specification>
```

Semantics:

**CLEAR STATION.** The station must be disabled prior to the clear request. If the station is waiting for a TERMINATE LOGICALACK, an acknowledgement is sent to the terminal. A recall-message request is sent, followed by a dummy output to the station to clear it.

If the station is in transaction mode at this time, it is taken out

**CLEAR PROGRAM.** The program must be disabled prior to the CLEAR request. If the program is a User or Port program, it must have been assigned a Common row, and the program must be busy. The program, is then DSed. If the program has a service and/or editor program not currently associated with another active program, the service and editor program are also DSed. When <stack designator> is <empty>, the single or master copy of the program is cleared. Otherwise, the program copy whose stack number matches the <stack designator> is cleared.

**CLEAR LINE.** A dial-in line that is connected and whose station is not busy is disconnected.

## DISABLE Request

Syntax:

```
<disable request> ::=
    DISABLE <disable action>

<disable action> ::=
    <system option> UPDATES <service comment> /
    STATISTICS /
    <system option> ALL /
    <system option> ALL
    <inclusive identification list> /
    <specific identification list> /
    <specific identification list> SERVICE
    <service comment>
```

<system option> ::=  
    <system specification> / <empty>

## Semantics:

A <disable request> has essentially the opposite effect of an <enable request> except that System Network Controllers cannot be disabled. If a non-busy program is disabled, the MCS will tell it to go to end of task; if a busy program is disabled, the MCS will wait until it is not busy (i.e., until it finishes its current transaction), and then tell it to go to end of task. When <system option> is <empty>, the request affects all systems. Otherwise, only the specified system is affected. (For the use of the word SERVICE, refer to the discussion on System Service Messages in section 8.)

### Example 1:

```
MESSAGE:  ?DISABLE ALL PROGRAMS
RESPONSE: # TTTT SYS1(1) PROGRAMS
          ALL DISABLED
```

### Example 2:

```
MESSAGE:  ?DISABLE ALL TEST
RESPONSE: # TTTT SYS1(1) TEST ALL
          DISABLED
```

### Example 3:

```
MESSAGE:  ?DISABLE I(2)
RESPONSE: # TTTT INPUTQUEUE (2)
          DISABLED
```

### Example 4:

```
MESSAGE:  ?DISABLE LINE (0:0:1)
RESPONSE: # TTTT LINE (0:00:01)
          DISABLED
```

### Example 5:

```
MESSAGE:  ?DISABLE UPDATES
RESPONSE: # TTTT ALL SYSTEMS
          UPDATES ALL DISABLED
          $.4. MGTESTSYS1 SYSTEM
          QUERYS AND ADMIN
          ONLY UNTIL FURTHER
          NOTICE MM/DD/YY TTTT
```

## DUMP Request

Syntax:

```
<dump request> ::= DUMP DATACOM
```

Semantics:

DUMP DATACOM initiates an on-line Datacom (Data Communications) queue file dump (CONTINUOUSPROCESSING must have been set TRUE in the TCL GLOBAL section). Records in the Datacom queue files are dumped to an archive tape and then returned to the system for reuse. For more information, refer to section 9.

Example:

```
MESSAGE:  ?DUMP DATACOM
RESPONSE: DATACOM WILL BE DUMPED
```

## ENABLE Request

Syntax:

```
<enable request> ::=
    ENABLE <enable action>

<enable action> ::=
    <system option> UPDATES <service comment> /
    STATISTICS /
    <system option> ALL /
    <system option> ALL
    <inclusive identification list> /
    <specific identification list> /
    <specific identification list> SERVICE
    <service comment>
```

Semantics:

ENABLE UPDATES. A DISABLE UPDATES Network Control command causes all transactions to be denied to programs specified as MODIFY = TRUE in their TCL declaration. ENABLE UPDATES allows transactions to be resumed for these programs. (For the use of the word SERVICE refer to the discussion on System Service Messages in this section.)

ENABLE STATISTICS. ENABLE STATISTICS allows the accumulation of run-time statistics for the system.

ENABLE <system option> ALL. ENABLE ALL readies all lines in the network and enables input for all stations whose system is not recovering. All input queues and programs whose system is not recovering, and which were not previously disabled by a Network Control command, are made operative. If previously disabled by a Network Control command, Input queues, stations, and programs can only be enabled by a specific enable request. When <system option> is <empty>, the request affects all systems. Otherwise, only the specified system is affected.

ENABLE <system option> ALL <inclusive identification list>. When <system option> is <empty>, the request affects all systems. Otherwise, only the specified system is affected.

STATIONS, TEST, and SPECIAL. If the system is not recovering, all stations are input enabled. If TEST and SPECIAL are specified, those stations with TEST = TRUE or SPECIAL = TRUE respectively are enabled.

INPUTQUEUES and PROGRAMS. If the system is not recovering and was not previously disabled by a Network Control command, all Input queues and programs are made operative.

LINES. All lines are made ready.

COMMONS. All commons are made operative.

SYSTEMSPOS and SYSTEMMONITORS. All System Network Controllers and System Monitors are input enabled.

ENABLE <specific identification list>. This readies any one of the items of the <specific identification list>. Following is a discussion of each member of this list.

1. STATION – The station is input-enabled.
2. INPUTQUEUE and PROGRAM – If previously disabled and still busy, a CLEAR request is required first for a program. Otherwise, if the system is not recovering, the Input queue and the program are made operative. When <stack designator> in a program specification is <empty>, the single or master copy of the program is affected. Otherwise the program copy whose stack number is equal to the <stack designator> is affected.
3. AREA – If the system is not recovering, all stations in the area are input-enabled.
4. LINE – The line is made ready. If the system is not recovering, all stations on the line are input-enabled.
5. COMMON – The Common area is made operative.
6. DCP – All lines on the DCP are made ready. If the system is not recovering, all stations on the lines are input-enabled.
7. CLUSTER – All lines on the cluster are made ready. If the system is not recovering, all stations on the lines are input-enabled.

Example 1:

```
MESSAGE:  ?ENABLE UPDATES
RESPONSE: # TTTT All SYSTEMS
          UPDATES ALL ENABLED
          $.5. MGTESTSYS1 SYSTEM
          RESUME FULL
          OPERATIONS MM/DD/YY
          TTTT
```

Example 2:

```
MESSAGE:  ?ENABLE ALL
RESPONSE: #TTTT SYS1 (1) ALL ENABLED
```

Example 3:

```
MESSAGE:  ?ENABLE ALL PROGRAMS
RESPONSE: #TTTT SYS1 (1) PROGRAMS
          ALL ENABLED
```

Example 4:

```
MESSAGE:  ?ENABLE S(101)
RESPONSE: #TTTT STATION (101)
          ENABLED
```

Example 5:

```
MESSAGE:  ?ENABLE CLUSTER (0:1)
RESPONSE: #TTTT CLUSTER (0:01)
          ENABLED
```

## INTERROGATE Request

Syntax:

```
<interrogate request> ::=
    INTERROGATE <interrogate identification list> /
    INTERROGATELOCAL <interrogate identification list>
```

```
<interrogate identification list> ::=
    <interrogate identification> /
    <interrogate identification list> ,
    <interrogate identification>
```

<interrogate identification> ::=  
    <station specification> /  
    <line specification>

Semantics:

An interrogate station environment is sent to request NDL-related data communications information. The following information is returned:

STATION NAME:

STATION IS ENABLED (EIP), STATION IS READY (RDY),  
STATION IS ATTACHED (ATT),  
RECONFIGURATION IN PROCESS (CIP),  
STATION IS CAPABLE OF AUTOMATIC SEQ MODE (SEQ),  
NDL SPECIFIED RETRY COUNT (RETRY =)  
MCS NUMBER AND STATION LSN IF STATION  
BELONGS TO ANOTHER MCS

DCP STATION INFO:

STATION PRIORITY (PRI =), STATION ACKNOWLEDGE (ACK),  
STATION IS QUEUED (QUD), STATION IS ENABLED (EIP),  
STATION IS READY (RDY), STATION TALLY (TALLY =),  
STATION IS VALID (VALID),

LAST FLAG SET IN NDL (LAST FLAG =),  
NDL STATION FLAGS (FLAGS =),

RECEIVE ADDRESS CHARS IF ANY (TSC =),  
TRANSMIT ADDRESS CHARS IF ANY (CDC =),

STATION TOGGLES (TOGS =), RETRIES LEFT (RETRY LEFT =),  
RETRIES (RETRY =), TALLY 0, TALLY 1, TALLY 2

TERMINAL INFO:

TERMINAL IS A SCREEN DEVICE (CRT)  
MYUSE = (1 = IN, 2 = OUT, 3 = IN/OUT)  
TERMINAL LINE WIDTH (WIDTH =)  
TERMINAL BUFFER SIZE (BUFFER/MAXINPUT =)  
TERMINAL PAGE SIZE (PAGE =)  
TERMINAL MAX PAGE SIZE (MAX PAGE =)

LINE NUMBER:

LINE IS READY (RDY), RECONFIGURATION IN  
PROGRESS (CIP),  
DISCONNECT ABORTED (DIS), SWITCHED LINE  
ERROR ENCOUNTERED  
(SLE), PHONE RINGING (RNG), LINE STATUS  
CURRENTLY CHANGING  
(SUB), LINE IS CONNECTED (CON), AUTO-ANSWER  
INFORCE (ANS),  
LINE HAS AUTOMATIC CALLING UNIT: MAY BE

DIALED OUT (ACU),  
LINE IS A SWITCHED LINE (SWT),  
CURRENT NUMBER OF STATIONS ON THIS LINE (STA/LINE =),  
MAX STATIONS ALLOWED ON LINE (MAXSTATIONS =)

DCP LINE INFO:

LINE IS PRIMARY [FULL DUPLEX ONLY] (PRI),  
LINE IS SECONDARY [FULL DUPLEX ONLY] (SEC),  
LINE CONTROL INDEX (CNT INDEX =), LINE NOT READY  
PENDING (NRP),  
SWITCHED LINE ERROR (SLE), PHONE RINGING (RNG),  
BUSY IN SWITCHED REQUEST (SWB), LINE IS CONNECTED (CON),  
AUTOANSWER IN FORCE (ANS), LINE HAS ASSOCIATED ACU (ACU),  
LINE IS A SWITCHED LINE (SWT), LINE HAS CONTROLLED  
CARRIER (CCA),  
LINE IS BUSY (BSY), LINE NOT TO BE USED,  
SYNCHRONOUS LINE (SYNC)

LINE IS READY (RDY), LINE IS BUSY (BSY), LINE WRITE READY  
(WRY)  
LINE ACKNOWLEDGE READY (ACK), LINE IS CONNECTED (CON),  
LINE IS QUEUED (QUD), CURRENT STATION NUMBER (STATION =),  
MAX STATIONS ON LINE (MAXSTATIONS =)

LINE TOGGLE [1] (TG1 =), LINE TOGGLE [0] (TG0 =),  
LINE TALLY [1], (TLY1 =), LINE TALLY [0] (TLY0 =),  
INDEX FOR COLINE (COLIN# =),

ADAPTER TYPE (TYPE =) SYNC/ASYNCR, BPS, CHAR, FRAME

INITIATE TRANSMIT DELAY, INITIATE RECEIVING DELAY,  
TIMEOUT FOR THIS LINE

If line has automatic calling unit:

INDEX FOR ASSOCIATED ACU, INDEX FOR CONTINUE

The INTERROGATELOCAL request generates a similar response to the INTERROGATE request; however, it uses the 104 DCWRITE to get the local DCP table information. This would be useful if the local table information had changed but the main-memory tables had not been updated.

INTERROGATELOCAL gives some of the same information as the response to the ?INTERROGATE command, as well as the following additional information:

STATION INFO:

STATION FREQUENCY (FREQ)  
REQUEST SET NUMBER (REQUEST SET)

LINE INFO:

INDEX REGISTER FROM CLUSTER (IR)  
BC/BI REGISTER FROM CLUSTER (BCBI)



Example 1:

MESSAGE: ?INTERROGATE S TTY2  
RESPONSE:

STATION TTY2:EIP,RDY,ATT,,SEQ,WIDTH = 72,RETRY = 15  
BELONGS TO MCS (1), LSN = 4  
DCP STATION INFO:PRI = 0,,,EIP,RDY,TALLY = 0, VALID  
LAST FLAG = 255,FLAGS = 0000000000000000000000  
TOGS = 00000000 ,RETRY LEFT = 15, RETRY = 15  
TLYO = 0,TLY1 = 0,TLY2 = 0  
TERMINAL INFO:MYUSE = 3, WIDTH = 72, BUFFER/MAXINPUT = 80  
PAGE = 0, MAXPAGE = 0  
LINE(00:00:11):RDY,,,,,,,,STA/LINE=1, MAXSTATIONS = 1  
DCP LINE INFO:,,CNT INDX = 1,,,,,,,,,  
RDY,,,CON,,STATION = 0,MAXSTATIONS = 1  
TG1 = 0,TGO = 0,TLY1 = 0,TLYO = 0,COLIN# = 0  
TYPE=10100 ASYNC,110 BPS,8-BIT CHAR, 11-BIT FRAME  
INIT TRANS DELAY = 0 ,INIT REC DELAY = 0  
TIMEOUT = 2.936 SEC

Example 2:

MESSAGE: ?INTERROGATELOCAL S(3)  
RESPONSE:

LOCAL DCP TABLES  
STATION RGEM6: EIP,RDY,,,  
FREQ = 0,REQUEST SET = 7,TALLY = 0,VALID  
TOGS = 00100100,RETRY = 5,INITIAL RETRY = 5  
TALLY[0] = 24,TALLEY[1] = 9,TALLY[2] = 1  
LINE (00:00:01)HALF DUPLEX,LINE CONTROL INDEX = 0  
:,,,,,,,,ASYNC,IR = 277,BCBI = 32  
LINESTATUS :BSY,,,CON,QUD  
TOG[1] = 1,TOG[0] = 0,STA INDEX = 2, MAX STATIONS = 17  
TALLY[1] = 8,TALLY[0] = 7

## MOVE Request

Syntax:

```
<move request> ::=  
  MOVE <station specification> TO  
  <line specification> <optional terminal name change>
```

Semantics:

The <move request> moves an NDL station from one line to another. The station must first be disabled. The <optional terminal name change> allows the station to be associated with a new terminal. This new terminal must be a valid NDL terminal name.

## RECOVER Request

Syntax:

<recover request> ::= RECOVER <system specification>

Semantics:

The <recover request> initiates recovery for the specified system. During recovery, the stations, input queues, and programs for the specified system are made inoperative, and enable requests are denied.

Example:

```
MESSAGE:  ?RECOVER SYSTEM (1)
RESPONSE: SYSTEM WILL BE
          RECOVERED
```

### CAUTION

Do not initiate recovery by Network Control command unless directed to do so by a system error message.

## RELEASE Request

Syntax:

<release request> ::=  
RELEASE <station designation> TO  
<MCS specification>

<MCS specification> ::=  
<identifier> / <identifier> ON <identifier>

Semantics:

The <release request> releases the specified station to the specified MCS.

Example:

```
? RELEASE STA1 TO SYSTEM/CANDE ON USERPACK
```

## RUN PROGRAM Request

Syntax:

<run program request> ::=  
RUN <no input program identification> /  
RUN <external program identification>

<no input program identification> ::=  
    <program number> / (<program number>)

<external program identification> ::=  
    <generalized identifier>

#### Semantics:

The <run program request> is used to run a "no-input" program. Details are contained in the PROGRAM section of section 3. The <program name> or <program number> must be defined in the TCL. The no-input program is processed as a task under the MCS.

The <run program request> may also be used to run external Object jobs. (Interface specifications are presented in section 4.) The Object job is processed as an independent task with a file called "HANDLER" label-equated to the originating station. If the internal name of the remote file of the Object job is not called HANDLER, then the <run program request> should not be used to initiate the Object job.

## STATUS Request

#### Syntax:

<status request> ::= STATUS <status action>

<status action> ::=  
    UPDATES /  
    STATISTICS /  
    ALL <inclusive identification list> /  
    <specific identification list>

#### Semantics:

STATUS UPDATES and STATUS STATISTICS. Whether these actions are enabled or disabled is indicated.

STATUS ALL <inclusive identification list>. The type of information returned depends on the <inclusive identification list>. Following is a discussion of each member of this list.

1. STATIONS. For each station that is not operative, the following information is returned delimited by commas:

STATION NAME AND NUMBER

DIALIN INDICATOR (DIN)

THIS STATION INTERCEPTING INDICATOR (IIF)

STATION INTERCEPTING THIS STATION (STATION NUMBER OF INTERCEPTING STATION)

OUT OF SERVICE VIA EXCESSIVE NDL ERRORS (SUCH AS TIMEOUT OR PARITY) OR \$ SERVICE MESSAGE (OOS)

NAK ON SELECT INDICATOR (NOS)

DISABLED BY NETWORK CONTROL (MAN)  
 NOT OPERATIVE INDICATOR (NOP)  
 STATION IS IN TERMINATE LOGICAL ACK WAITING FOR PROGRAM TO ACK  
 (PAC)  
 STATION BUSY INDICATOR (BSY)  
 SUCCESSFUL ENABLE INPUT INDICATOR (EIP)  
 OUTPUT READY INDICATOR : BOT NOT = TOP (RDY)  
 OUTPUT QUEUED INDICATOR (QED)  
 WAITING FOR PRIMARY OUTPUT INDICATOR (WTG)  
 RECEIVED INPUT FROM STATION BUT NO ASSOCIATED OUTPUT YET:  
 TRANSACTION STATE (RCV)  
 STATION IS IN PAGING MODE (PGM)  
 STATION BOT-TOP  
 NUMBER OF OUTPUT MESSAGES QUEUED  
 BLOCKED OUTPUT IS CURRENTLY BEING TRANSMITTED  
 TO THIS STATION (PAG) CHAR POSITION OF  
 BLOCK BREAK (integer) NEXT RECORD TO  
 PROCESS (integer)

2. INPUTQUEUES. For each Input queue that is not operative, the following information is returned delimited by commas.

INPUTQUEUE NAME AND NUMBER

SPECIFIC DISABLE BY NETWORK CONTROL (MAN)  
 NOT OPERATIVE INDICATOR (NOP)  
 QUEUE INSERT ALLOWED; MEMORY LIMIT NOT EXCEEDED (QIN)  
 VALUE OF RERUN STATE (RERUN)  
 READY INDICATOR: BOT NOT = TOP (RDY)  
 WAITING FOR ANOTHER COPY OF THE PROGRAM TO BE INITIATED (WTG)  
 BUSY: TRANSACTION IS BEING PROCESSED OUT OF THIS QUEUE (BSY)  
 INPUTQUEUE BOT-TOP  
 NUMBER OF UNPROCESSED MESSAGES

3. PROGRAMS. For each program that is not operative, the following information is returned delimited by commas.

PROGRAM NAME AND NUMBER

DISABLED BY NETWORK CONTROL (MAN)  
 NOT OPERATIVE INDICATOR (NOP)  
 COMMON ASSIGNED INDICATOR (OPN)  
 BUSY: PROGRAM IS PROCESSING A TRANSACTION (BSY)  
 SOME STATION IS HOLDING UP OUTPUT WAITING FOR THIS PROGRAM TO  
 GENERATE A PRIMARY OUTPUT (HLD)  
 READY: RUNNING AND IN THE MIX (RDY)  
 NUMBER OF PROGRAMS CURRENTLY SERVICED BY THIS SERVICE OR  
 EDITOR PROGRAM (RUN =)  
 NUMBER OF COPIES OF THIS PROGRAM CURRENTLY ACTIVE  
 (ACT =)  
 STATUS  
 PROCESSOR TIME  
 I/O TIME  
 ELAPSED TIME

4. LINES. For each line, the status of all stations on the line is returned, followed by the following information delimited by commas.

DCP:CLUSTER:ADAPTER

NOT OPERATIVE INDICATOR (NOP)  
READY: AT LEAST 1 STATION ON LINE IS RDY (RDY)  
LINE SWITCH ERROR INDICATOR (SLE)  
RINGING INDICATOR (RNG)  
SWITCH BUSY INDICATOR (SWB)  
CONNECTED INDICATOR (CON)  
AUTO ANSWER INDICATOR (ANS)  
DIALOUT INDICATOR (ACU) IIIALIN INDICATOR (SWT)  
CURRENT STATION NUMBER USING THE LINE  
NUMBER OF STATIONS ON THE LINE  
WAITING TERMINATE NORMAL INDICATOR (ACK)

5. COMMONS. For each Common row, the following information is returned delimited by commas.

ROW NUMBER

NOT OPERATIVE INDICATOR (NOP)  
READY: PROGRAM ASSIGNED TO COMMON  
BUSY: PROCESSING A TRANSACTION OUT OF THIS COMMON (BSY)  
INPUTQUEUE NAME AND NUMBER  
PROGRAM NAME AND NUMBER  
COMMON[COM,0]  
COMMON[COM,1]

6. TEST. As previously described, for each test station that is not operative, a station status is returned.
7. SPECIAL. As previously described, for each special station that is not operative or which has output queued, a station status is returned.
8. ENABLED. As previously described, for each enabled station that is not operative, a station status is returned.
9. SYSTEMSPOS or SYSTEMMONITORS. As previously described, for each station that is a SYSTEMSPO, a station status is returned; or for each SYSTEMMONITOR, a station status is returned.
10. MCSS. For each MCS the name, number, and running status is returned.

STATUS <specific identification list>. The information returned, as described in the semantics for STATUS <inclusive list>, also applies for STATUS <specific identification list>. The status is returned regardless of whether the object of the status is operative. In addition, STATUS DCP and STATUS CLUSTER may be requested and a status for all lines on the respective object is returned. STATUS MKE and STATUS MID may be requested, and information about the specified message key or output message id is returned. On a program status, when <stack designator> is <empty>, a status for the single or master copy of the program is returned. Otherwise, a status for the program whose stack number is equal to the <stack designator> is returned.

Example 1:

MESSAGE: ?STATUS ALL STATIONS  
RESPONSE: TTY2(1),,,,,,MAN,NOP,,,,RDY,,,2-  
58,4  
TD14(2),,,,,,MAN,NOP,,,,,3-3,0  
TD12(4),,,,,,MAN,NOP,,,,,5-5,0  
CONO1(5),,,,,,MAN,NOP,,,,,6-  
6,0  
TD31(7),,,,,,MAN,NOP,,,,,8-8,0  
#TTTT ALL STATIONS  
STATUS

Example 2:

MESSAGE: ?STATUS I(13)  
RESPONSE: ECHOTEST13(13),,,QIN,,RDY,,84-  
86,1

Example 3:

MESSAGE: ?STATUS CLUSTER(0:0)  
RESPONSE: TTY3(3),,,,,,,,,EIP,,,68-68,0  
0:00:12,,RDY,,,,,,,,(3),  
TD13(6),,,,,,,,,EIP,,,7-7,0  
0:00:10,,,,,,,,(6),

Example 4:

MESSAGE: ?STATUS P ECHOTEST1  
RESPONSE: ECHOTEST1(5),,,OPN,,RDY  
RUN = 1, ACT = 1:SLEEPING  
  
PROC = 00:00:00 I/O = 00:00:00  
ELAP = 00:08:41

Example 5:

MESSAGE: ?STATUS C(19)  
RESPONSE: 19,,BSY, IQU:ECHO(13)  
PRO:ECHO(5) ,  
C[0] = 1 , C[1] = 8

Example 6:

MESSAGE: ?STATUS MKE ECHOIT  
RESPONSE: MESSAGEKEY ECHOIT  
PROGRAM TP1(1)  
INPUTQUEUE IQU1(1)  
MODULE 0  
FUNCTION 0

Example 7:

```
MESSAGE:  ?STATUS MID FMT1
RESPONSE: MESSAGEID FMT1
          FORMAT F8 DEVICE(1)
```

## SUBTRACT Request

Syntax:

```
<subtract request> ::=
    SUBTRACT <station specification>
```

Semantics:

The <subtract request> removes a station from a line. The station must first be disabled.

## SYSTEM Request

Syntax:

```
<system request> ::= SYSTEM <system action>
```

```
<system action> ::=
    <EOJ type> EOJ <service comment> /
    VERSION / STATUS
```

```
<EOJ type> ::= FAST / CLEANUP / <empty>
```

```
<service comment> ::= <string> / <empty>
```

Semantics:

The <system action> EOJ disables inputs on all stations in the on-line environment and sends the MCS to End-of-Job. If the <EOJ type> is CLEANUP, the MCS terminates after all queued input is processed and all associated output is delivered.

### NOTE

A NOINPUT program could delay CLEANUP EOJ indefinitely.

If the <EOJ type> is <empty>, the MCS terminates when the current input for each program is completed and all outputs in process are delivered (i.e., as each program becomes NOT BUSY, it is deprived of a new input to process even if there is one queued for it). If <EOJ type> is FAST, the action is the same as though it were <empty>, except that the MCS terminates a program that timeouts. The <system action> VERSION sends the current system level to the station. The <system action> STATUS sends run-time information to the station.

Example 1:

MESSAGE: ?SYSTEM STATUS  
RESPONSE:

SYSTEM STATUS: RUNNING  
5 STATIONS NOT OPERATIVE.  
0 INPUTQUEUES NOT OPERATIVE.  
0 STATIONS PROCESSING OUTPUT  
1 PROGRAM PROCESSING INPUT  
0 OBJECT PROGRAMS RUNNING.  
0 STATIONS BEING MONITORED.  
0 STATISTICS ENABLED.  
0 SYSTEMS UPDATES DISABLED.

Example 2:

MESSAGE: ?SYSTEM VERSION  
RESPONSE:

CONTROL VERSION 04.000.000 THURSDAY 09/20/79 1531 RELEASE  
400 DCP/BLUE0 DATACOM PROCESSOR CODE 09/23/79 0354

Example 3:

MESSAGE: ?SYSTEM EOJ  
RESPONSE:

JOB 8063 TASKS 8064,8071,8071,8069 09/25/79 1255  
PROCESS TIME 00:00:15 00 PERCENT OF ELAPSED  
IO TIME 00:00:11 00 PERCENT OF ELAPSED  
ELAPSED TIME 00:30:15  
CONTROL(0) TOTAL MSGS = 4 AVERAGE ELAP = 0.113 (0.106) SEC.  
AVERAGE MSGS/HOUR SINCE 1252 WAS 86  
INPUT RECS = 41 , ACKS = 0 , MSG REUSE = 0  
OUTPUT RECS = 53 , MSG REUSE = 106  
SIZE8QUEUE MSGCOUNT = 7 SIZE = 56 WORDS.  
\* SYSTEM END OF JOB 09/25/79 1255



## TABLE Request

Syntax:

<table request> ::= TABLE <table action>

<table action> ::=  
STATISTICS /  
<table identification list>

<table identification list> ::=  
<table identification> /  
<table identification list> , <table identification>

<table identification> ::=  
<station specification> /  
<input queue specification> /  
<program specification> /  
<line specification> /  
<MCS specification>

Semantics:

TABLE STATION. A table is returned containing the following information.

STATION NAME (STATION NUMBER) LSN DLS BOTBOT  
IN MSGS, OUT MSGS, BACKWARD LINK IN STATION INPUT QUEUE  
WIDTH, PAGES PER BUFFER  
POLL FREQ STATION IS CRT (CRT),  
STATION RECEIVES BLOCKED INPUT (BLK),  
STATION ACK INITIATED BY PROGRAM (PROACK),  
STATION IS MONITOR (MON),  
STATION IS NETWORK CONTROL (SPO),  
STATION CAN INTERCEPT (INT),  
STATION RECEIVES BROADCAST (BDC),  
STATION RECEIVES SERVICE MESSAGES (SER),  
STATION DOES NOT RECEIVE TITLE (NOT),  
STATION IS MASTER (MAS),  
STATION USES ROUTING HEADER (ROU),  
STATION IS SPECIAL (SPC),  
STATION CAN OPERATE IN TEST (TST),  
STATION DOES NOT ORIGINATE MSG  
(COMP TO COMP) (ORG)  
STATION DOES NOT QUEUE OUTPUT (NQU)  
APPLICATION (REQUEST SET), LOGICAL ACK, MYUSE  
SYNC/ASYN, BPS, CHAR, FRAME  
MODE

STATION SYSTEM NAME (STASYS)  
ALTERNATE STATION  
ALTERNATE TO STATION  
STATION BITS  
AREAS

TABLE PROGRAM. When <stack designator> is not <empty>, the table returned is for the single or master copy of the program. Otherwise, the table is for the copy whose stack number matches the stack number in the <stack designator>. The table contains the following information:

PROGRAM NAME (PROGRAM NUMBER) TITLE  
NUMBER OF STATIONS ASSIGNED TO IF ANY  
NUMBER OF SUBSPACES DECLARED PRIORITY  
NUMBER OF COPIES RUNNING (IF NOT SERVICE OR EDITOR)  
STACK NUMBERS OF COPIES RUNNING  
INPUTQUEUES  
TYPE PERMANENT (PER), MULTIPLE (REN), NO INPUT (INP),  
MODIFY (MOD) CONTROLBIT  
SERVICE PROGRAM IF USED  
EDITOR PROGRAM IF USED  
INACTIVE TIMEOUT IF SPECIFIED  
MAXCOPIES  
TIMEOUT

TABLE INPUTQUEUE. A table is returned containing the following information:

INPUTQUEUE NAME (INPUTQUEUE NUMBER)  
DYNAMIC VOLUME CONTROL QUEUE DEPTH (QUD)  
DYNAMIC VOLUME CONTROL TIME LIMIT (RTL)  
LIMIT OF MEMORY RESIDENT MESSAGES: MEMORYLIMIT (LMT)  
RECOVERY SPECIFIED (RCV),  
WAIT FOR AUDIT (ADT)  
NUMBER OF MESSAGES PASSED THROUGH THE QUEUE (QNM)  
MESSAGES/DAY  
RECOVERY BOT (RCVBOT)  
BOTBOT  
STATISTICS IF STATISTICS ARE ENABLED  
PROGRAM NAME ( PROGRAM NUMBER)

TABLE STATISTICS. If statistics are enabled, a table of statistics is returned.

TABLE MCS. The MCS name and number are returned along with a run status (whether initiated, running, DSed, or required).

Example 1:

MESSAGE: ?TABLE STATISTICS  
RESPONSE:

JOB 8063 TASKS 8064,8070,8071,8069, 01/09/76 0431  
PROCESS TIME 00:00:04 00 PERCENT OF ELAPSED  
IO TIME 00:00:07 00 PERCENT OF ELAPSED  
ELAPSED TIME 00:19:15  
CONTROL(0) TOTAL MSGS = 2 AVERAGE ELAP = 1.466 (0.816) SEC.  
AVERAGE MSGS/HOUR SINCE 1301 WAS 271  
INPUT RECS = 3 , ACKS = 0 , MSG REUSE = 0  
OUTPUT RECS = 3 , MSG REUSE = 2  
SIZE8QUEUE MSGCOUNT = 4 SIZE = 32 WORDS.

Example 2:

MESSAGE: ?TABLE P(5)  
RESPONSE:

ECHOTEST1(5) TITLE = MG/TEST/PROG/1  
1 COPY RUNNING  
STACK NUMBER IS : 8074.  
INPUTQUEUES : ECHOTEST13(13)  
USER PROGRAM PER,REN,, CONTROL BIT = 0  
EDITOR : EDITOR1(1) CONTROL BIT = 1

Example 3:

MESSAGE: ?TABLE S TTY3  
RESPONSE:

TTY3(3) LSN = 5 DLS = 0:0:12:0 BOTBOT = 4  
IN MSG/DAY = 34,OT MSG/DAY = 35, STA IN LINK = 40  
WIDTH = 72,PAGE/BUFFER = 65535  
POLL FREQ = 255,,,MON,SPO,INT,,SER,NOT,,,,,  
APPLICATION = 1,LOGICALACK = 2,MYUSE = 3  
AYSNC,110 BPS,8-BIT CHAR,11-BIT FRAME  
STATION IN DATA MODE  
STATION SYSTEM NAME : MTESTSYS1  
ALTERNATE : CONO1(5)  
ALTERNATE TO : TD12(4)  
AREAS: NONSCREENS(2) MIXEDTYPE(3)

Example 4:

MESSAGE: ?TABLE I(13)  
RESPONSE:

ECHOTEST13(13) QUD=1 RTL=1 LMT=255  
RCV,ADT QNM=0 MSGS/DAY=3 RCVBOT=15 BOTBOT=5  
ECHOTEST1(5)

## UPDATE Request

Syntax:

```
<update request> ::=
    UPDATE <update identification list>

<update identification list> ::=
    <update identification> /
    <update identification list> , <update identification>

<update identification> ::=
    <program specification> /
    <input queue specification> /
    <station specification> /
    <format specification> /
    <function specification> /
    <access key specification> /
    <message key specification> /
    <valid users specification> /
    <device specification>

<format specification> ::=
    FORMAT <format designator list>

<format designator list> ::=
    <format designator> /
    <format designator list> , <format designator>

<function specification> ::=
    FUNCTION <function designator list>

<function designator list> ::=
    <function designator> /
    <function designator list> , <function designator>

<access key specification> ::=
    <system specification> ACCESSKEY <access key> =
    <access key MKE list>

<access key> ::= <identifier>

<access key MKE list> ::=
    <access key MKE> /
    <access key MKE list> , <access key MKE>
```

```

<access key MKE> ::=
    <optional delete char> ALL /
    <optional delete char> <message key>

<optional delete char> ::= - / <empty>

<message key> ::= <identifier>

<message key specification> ::=
    <system specification> MKE <message key>
    <message key attributes>

<message key attributes> ::=
    DELETE / <message key attribute list>

<message key attribute list> ::=
    <message key attribute> /
    <message key attribute list>, <message key attribute>

<message key attribute> ::=
    <input queue attribute> / <mod fun attribute>

<input queue attribute> ::=
    I = <input queue designator> /

INPUTQUEUE ::=
    <input queue designator>

<mod fun attribute> ::=
    MODFUN = (<integer>, <integer>)

<valid users specification> ::=
    <access key specification> /
    VALIDUSERS <station name> =
    <access key specifications list>

<access key specification list> ::=
    <access key specification> /
    <access key specification list>,
    <access key specification>

<access key specification> ::=
    <optional delete char> ALL
    <optional delete char> <access key>

```

```

<device specification> ::=
    <system specification> DEVICE (<device number>)
    <format association>

```

```

<device number> ::=
    <integer>

```

```

<format association> ::=
    FORMATSIN: <format name> =
        <formatsin MKE list> /
    FORMATSOUT: <format name> =
        <formatsout MID list> /
    NONFORMATTED: <formatsin MKE list>

```

```

<format name> ::= <identifier> / DELETE

```

```

<formats in MKE list> ::=
    <formats in MKE> /
    <formats in MKE list> , <formats in MKE>

```

```

<formats in MKE> ::=
    <message key> <optional item count>

```

```

<optional item count> ::= (<integer>) / <empty>

```

```

<formats out MID list> ::=
    <message id> /
    <formats out MID list> , <message id>

```

```

<message id> ::= <identifier>

```

#### Semantics:

UPDATE PROGRAM, INPUTQUEUE, OR STATION. The update request causes the memory tables for the specified <update identification> to be backed up to the system control file.

UPDATE FORMAT. UPDATE FORMAT causes the formats listed to be replaced by the current patch version of the formats for all stations. Prior to the request, patched formats can only be "seen" by stations in practice mode. Thus, formats may be tested first on a station in practice mode, and then made permanent for the rest of the system. A detailed discussion of UPDATEFMT is contained in the CONTROL section of the TCL syntax in section 3.

UPDATE FUNCTION. UPDATE FUNCTION causes the functions listed to be replaced by the current patch version of the function. Prior to this request, patched functions can only be "seen" by stations in practice mode. A detailed discussion of UPDATEFMT is contained in the <control section> of the TCL syntax in section 3.

**UPDATE ACCESSKEY.** UPDATE ACCESSKEY allows for adding or deleting access keys or modifying their associated message key list. If the access key named cannot be found, it is assumed to be new. Each message key in the <access key mke list> that does not appear with an <optional delete char> is added to the list of message keys for the access key. Those with an <optional delete char> are deleted from the access key. If -ALL is specified, the access key is deleted. The number of access keys added must fall within the MAXNEWACCESSKEYS value specified in the <global section> of the most recent TCL generation. Each access key added decreases this value by one, and deleting an access key does not increase this value. Updates are backed up to the system control file so that they are maintained across MCS EOJs.

**UPDATE MESSAGE KEY.** UPDATE MKE allows for adding or deleting message keys or changing attributes for existing message keys. If the message key cannot be found, it is assumed to be a new message key, and in this case, the Input queue attribute must appear first. For existing message keys, the Input queue may be changed along with MODFUN values. The number of message keys added must fall within the MAXNEWMKES value specified in the GLOBAL Section of the most recent TCL generation. Each message key added decreases this value by one, and deleting a message key does not increase this value. Updates are backed up to the system control file so that they are maintained across MCS EOJs.

**UPDATE VALIDUSERS.** UPDATE VALIDUSERS allows the valid user list of a station to be modified by adding or deleting access keys. Updates are backed up to the system control file so that they are maintained across MCS EOJs.

**UPDATE DEVICE.** UPDATE DEVICE allows for a restatement of the message key to format association for a given device. FORMATSIN names the message key associated with a format for the given device; FORMATSOUT names the message-IDs associated with a format for a given device; and NONFORMATTED names message keys with no format on the given device. If DELETE is used instead of a format name, the format is deleted for that device. Updates are backed up to the system control file so that they are maintained across system EOJs.

Example 1:

```
MESSAGE:  ?UPDATE P(5)
RESPONSE: # TTTT PROGRAM (5)
          UPDATED
```

Example 2:

```
MESSAGE:  ?UPDATE S TD31
RESPONSE: # TTTT STATION TD31
          UPDATED
```

Example 3:

```
MESSAGE:  ?UPDATE I(13)
RESPONSE: # TTTT INPUTQUEUE (13)
          UPDATED
```

## WHERE Request

Syntax:

```
<where request> ::=  
    WHERE <system option> <where option>
```

```
<where option> ::=  
    <station specification> /  
    <access key> /  
    <empty>
```

Semantics:

The <where request> returns which access keys are logged on at which "SIGNON = TRUE" stations.

Example 1:

User not logged on:

```
MESSAGE:  ? WHERE SYSTEM (1) USER1  
RESPONSE: # USER1 NOT ON SYSTEM (1)
```

Example 2:

User1 logged on at STA1 in SYSTEM (1)

```
MESSAGE:  ?WHERE SYSTEM(1) USER(1)  
RESPONSE: #STA1(22): USER1
```

## ZIP Request

Syntax:

```
<zip request> ::= ZIP <string>
```

Semantics:

The <zip request> allows the operator at a System Network Control station to initiate input to the work flow language compiler via the MCP. The contents of the <string> are placed in an array row and ZIP-executed. The MCS performs no syntax checking on the contents of the <string>.



## SYSTEM ERROR MESSAGES

During the running of the Data Communications System, errors may occur which require operator intervention. Some actions that produce errors are retried without operator intervention, but some are so severe that operator intervention is the only solution. Messages indicating the errors are sent to the Network Control stations for operator action. System error messages are generated indicating errors in different sections of the Data Communications System. The sections are data communication errors, input message processing errors, output message processing errors, and transaction processing program errors.

Data communication errors are those errors occurring prior to a message being received by the MCS or after a message is returned. They normally indicate an error associated with a station, line, or data communication interface.

Input message processing errors are those associated with the input disk file and occur when messages are read from disk.

Output message processing errors are those associated with the output disk file and occur when messages are read from disk.

Transaction processing program errors are those associated with the programs processing the messages. These messages indicate abnormal statuses of the programs.

### DATA COMMUNICATION ERROR Message

Syntax:

```
<data communication error message> ::=
  *ERR <error number>:<station name>(<station number>)
    :<time>:<status><error indicator list> /
  *ERR <switched line error number>:<station name>
    (<station number>):<time>:<switched line status>
    <switched status list>
```

```
<error number> ::=
  <DCP code error number> /
  <DCP hardware error number> /
  <no enable input space error number>
```

```
<DCP code error number> ::= 1 / 2 / 3
```

```
<DCP hardware error number> ::= 4 / 5 / 6
```

```
<no enable input space error number> ::= 7
```

```
<switched line error number> ::= 13 / 14
```

```
<time> ::= [time of error in military time]
```

```

<status> ::=
    INPUT / INPUT POLL / OUTPUT / OUTPUT SELECT /
    <empty>

<switched line status> ::=
    <completion status> DIALOUT: /
    <completion status> DISCONNECT: /
    <completion status> ANSWER THE PHONE: /
    INTERROGATE SWITCHED STATUS /
    <empty>

<completion status> ::=
    UNABLE TO INITIATE /
    UNABLE TO COMPLETE /
    <empty>

<error indicator list> ::=
    <error indicator> /
    <error indicator list> <error indicator>

<error indicator> ::=
    <timeout error indicator> /
    <stop bit error indicator> /
    <character buffer overflow error indicator> /
    <break error indicator> /
    <disconnect error indicator> /
    <vertical parity error indicator> /
    <horizontal parity error indicator> /
    <address error indicator> /
    <transmission number error indicator> /
    <format error indicator> /
    <output naked error indicator> /
    <end of buffer error indicator> /
    <loss of carrier error indicator>

<timeout error indicator> ::= TIMEOUT

<stop bit error indicator> ::= STPB

<character buffer overflow error indicator> ::= CBFO

<break error indicator> ::= BREAK

<disconnect error indicator> ::= DISCONNECT

<vertical parity error indicator> ::= PARITY

```

<horizontal parity error indicator> ::= HPARITY  
 <address error indicator> ::= AERR  
 <transmission number error indicator> ::= TERR  
 <format error indicator> ::= FERR  
 <output naked error indicator> ::= ONAK  
 <end of buffer error indicator> ::= EOB  
 <loss of carrier error indicator> ::= LOC  
 <switched status list> ::=  
     <switched status> /  
     <switched status list> < switched status>  
 <switched status> ::=  
     <switched error status> /  
     <ringing status> /  
     <switched busy status> /  
     <connected state status> /  
     <auto answer flag status> /  
     <dial out status> /  
     <dial in status>  
 <switched error status> ::= SWITCHEDERROR / <empty>  
 <ringing status> ::= RINGING / <empty>  
 <switched busy status> ::= SWITCHEDBUSY / <empty>  
 <connected state status> ::= CONNECTED / <empty>  
 <auto answer flag status> ::= AUTOANSWER / <empty>  
 <dial out status> ::= DIALOUT / <empty>  
 <dial in status> ::= DIALIN / <empty>

The <DCP code error numbers> and the <DCP hardware error numbers> are obtained from the result byte index and are documented in the DCALGOL manual, form number 5011430.

## INPUT MESSAGE PROCESSING ERROR Message

Syntax:

```
<input message processing error> ::=  
    *ERR 91* <input queue name> (<input queue number>)  
    :<time>:<input processing error>
```

```
<input processing error> ::= <input disk file parity> /  
    <input disk file end of file>  
    <input disk file bad linkage> /  
    <input disk file not ready>
```

```
<input disk file parity> ::= BAD INPUT QUEUE
```

```
<input disk file end of file> ::= BAD INPUT QUEUE
```

```
<input disk file bad linkage> ::= BAD INPUT QUEUE
```

```
<input disk file not ready> ::= BAD INPUT QUEUE -NOT
```

## OUTPUT MESSAGE PROCESSING ERROR Message

Syntax:

```
<output message processing error message> ::=  
    *ERR 90:<station name> (<station number>)  
    :<time>:<output processing error>
```

```
<output processing error> ::=  
    <output disk file parity> /  
    <output disk file end of file> /  
    <output disk file bad linkage> /  
    <output disk file not ready>
```

```
<output disk file parity> ::= BAD OUTPUT QUEUE
```

```
<output disk file end of file> ::= BAD OUTPUT QUEUE
```

```
<output disk file bad linkage> ::=  
    BAD OUTPUT QUEUE
```

```
<output disk file not ready> ::=  
    BAD OUTPUT QUEUE -NOT
```

## PROCESSING PROGRAM ERROR Message

Syntax:

```
<processing program error message> ::=
    <program identification part>
    <message identification part>
    <abort identification part>

<program identification part> ::=
    *PROGRAM <task status>:<time>:<mix index>
    <program title><crlf>
    INPUTQUEUE: <input queue name>
    (<program number>).<crlf>

<task status> ::=
    SUSPENDED / TIMEOUT / SCHEDULED /
    TERMINATED / NOT INITIALIZED

<mix index> ::= <integer>

<program title> ::= <generalized identifier>

<message identification part> ::=
    MESSAGE ADDRESS: <input queue base record address>
    , MESSAGE #: <system message number>
    , STATION: <station name>(<station number>).
    <crlf> / EMPTY

<system message number> ::= <integer>

<input queue base record address> ::= <integer>

<abort identification part> ::=
    <termination error identification> /
    <initialization error identification> / <empty>

<termination error identification> ::=
    <termination error> @ <terminal reference list>

<terminal reference list> ::=
    <terminal reference> /
    <terminal reference list> , <terminal reference>
```

<terminal reference> ::= <seg>:<pir>:<psr>

<seg> ::= [segment]

<pir> ::= [program instruction register]

<psr> ::= [program syllable register]

<termination error> ::=  
DIVIDE BY ZERO /  
EXPON OVERFLOW /  
INV INDEX /  
INTGR OVERFLOW /  
INACTIVE QUEUE /  
MEMORY PROTECT /  
INV OPERATOR /  
INSTR TIMEOUT /  
MEMORY PARITY /  
INV ADDRESS /  
STACK OVERFLOW /  
SEG ARRAY ERROR /  
SEQUENCE ERROR /  
INV PROG SYL / STACK UNDERFLOW

<initialization error identification> ::=  
NO CODE FILE ON DISK /  
NOT A PROGRAM FILE /  
NON EXECUTABLE CODE FILE /  
PARAMETER MISMATCH

<crlf> ::=  
[a sequence of characters to position to the left  
margin of a new line]

## System Service Messages

During the running of the Data Communications System, it sometimes is necessary to inform the stations what services are available to them. The information is passed to the stations by messages called System Service messages.

System Service messages are generated by certain System Control messages and inform a specified station and/or stations defined to receive them. Comments can be appended to standard service messages by supplying a <service comment>.

When an enable control message is issued, possible system-in-service System Service messages are generated. If the request is (ENABLE ALL <service comment>), all stations defined to receive service messages and not previously enabled are then sent a system-in-service message with the optional <service comment>. If the request is (ENABLE [station identification] SERVICE <service comment>), the specified station is sent an in-service-message. If the request is (ENABLE UPDATES <service comment>), all stations specified in TCL to receive service messages are sent an updates-in-service message.

When a disable control message is issued, possible system out-of-service System Service messages are generated. If the request is (DISABLE UPDATES <service comment>), then all stations specified in TCL to receive service messages are sent an updates-out-of-service message.

When a System Control message is issued telling the MCS to go to End-of-Job, a system out-of-service message is sent to all stations specified in TCL to receive service messages.

The general format for a service message is:

```
[STANDARD TITLE]    (OPTIONAL BY STATION DEFINITION)
$.<#>.<service text>
<service comment>    (OPTIONAL BY SYSTEM CONTROL MESSAGE)
```

Where:

```
<#> ::= SERVICE MESSAGE TYPE NUMBER
      1 (OUT OF SERVICE)
      2 (IN SERVICE)
      3 (COMMENTS ONLY)
      4 (UPDATES DISABLED)
      5 (UPDATES ENABLED)
```

```
<service text> ::=
  TYPE = 1 <system name> SYSTEM OUT OF SERVICE
  MM/DD/YY TTTT
  <station name> STATION OUT OF SERVICE
  MM/DD/YY TTTT
```

```
TYPE = 2 <system name> SYSTEM IN SERVICE
  MM/DD/YY TTTT
  STATION <station name> IN SERVICE
  MM/DD/YY TTTT
```

```
TYPE = 3 <empty>
```

```
TYPE = 4 <system name> SYSTEM QUERIES AND
  ADMIN ONLY UNTIL FURTHER NOTICE
  MM/DD/YY TTTT
```

```
TYPE = 5 <system name> SYSTEM RESUME FULL
  OPERATIONS MM/DD/YY TTTT
```

## SECTION 9

# CONTINUOUS PROCESSING

The B 5000/B 6000/B 7000 Series GEMCOS MCS can be run in a continuous-processing environment. The main objective here is to allow on-line dumps of the Data Communication (Datacom) queue files. In general, this means establishing a cut-off point in the station and Input queues and dumping all records prior to the cut-off point to an archive tape. After a successful archive, dumped records are made available for reuse in the queue files. The archive tape is labeled with the date and time of the cut-off point. This is in fact a control point in the data base to which the data base can be rebuilt. After a DMS II rebuild, GEMCOS archive tapes can be processed to resubmit archived transactions. This section discusses the areas of the system that are affected by running a continuous-processing MCS.

### Option Selection

There are two TCL options related to continuous processing: CONTINUOUSPROCESSING and QUEUEBLOCKSIZE.

CONTINUOUSPROCESSING is a Boolean option specified in the GLOBAL section of the TCL source deck. When it is set to TRUE, the MCS operates in a continuous-processing mode.

QUEUEBLOCKSIZE specifies the number of records preallocated for each Input queue and each station in the Datacom queue files. The default value is 50. This means that on the first continuous processing TCL generation, 50 records are allocated for each Input queue in the DATACOM/QUEUE/INPUT file, and 50 records are allocated for each station in the DATACOM/QUEUE/OUTPUT file. All records for a given Input queue or station are written to the same 50-record block. When a block becomes full, a search of an available block mask is made, and another 50-record block is allocated. When an on-line Datacom dump occurs, used records are dumped to an archive tape and then returned in blocks of 50 records to the available block mask. This allocation scheme cuts down the overhead of determining record availability at the individual record level.

### On-Line Datacom Dumps

On-line Datacom dumps are initiated by the DUMP DATACOM Network Control command. The sequence of events is as follows:

1. ENTER: ?DUMP DATACOM  
RESPONSE: DATACOM WILL BE DUMPED

The dump command is processed by the control stack. The only action is to set a Boolean to be picked up by Processeverything, which actually initiates the dump phases. The Boolean is written to a dump restart file and locked in the directory. The presence of this file on halt/load indicates that a dump was in progress and will be re-initiated at BOJ.

2. Processeverything disables all Input queues to establish a clean cut-off point for the dump.

MESSAGE: ALL SYSTEMS BEING DISABLED UNTIL ON-LINE DUMP STARTED



3. Process everything sends all TPs to EOJ and runs the restart TP for each system to delete the restart data set to insure that in the event of a data base abort there are no restart pointers that point logically before the cut-off point. All records prior to the cut-off points are dumped and may reside in blocks that are reused.
4. The current values of IQUBOTS and STABOTS are saved, and Input queues and programs are re-enabled.
5. An asynchronous dump stack is initiated. The Input queue and station cut-off points are immediately written to the dump restart file. Prior to this point, the dump restart file would point to phase 1. In the event of a restart, it would stop the network again and save new cut-off points. Now, however, the cut-off points are on the dump restart file, and in the event of restart, the dump process could immediately advance to this phase.

MESSAGE: ON-LINE DUMP INITIATED AND SYSTEM RE-ENABLED

6. The archival dump procedure is called from the dump stack. This is the same procedure that would be initiated as a result of an off-line TCL regeneration, thus insuring that archive tapes, whether created off-line or on-line, are the same format. The only difference between on-line and off-line dumps are the entries made in the DATACOM/LEDGER file. Archive tapes are logged in the ledger based on these factors:
  - a. Generation date and time
  - b. Sequence control number
  - c. Dump date and time

For off-line dumps, all of these factors change. For on-line dumps, generation date and time and sequence control number remain the same, and only dump date and time change.

At the end of the archive procedure, the operator is required to indicate whether the tape should be logged in the ledger. If NO is entered, the entry in the DATACOM/LEDGER file is eliminated and the whole dump is invalidated. The beginnings of the queues (BOTBOTS) are not advanced, and no records are returned for reuse. If YES is entered, the tape is logged, BOTBOTS are advanced to the last record dumped for each queue, and dumped records are reused. For a queue in error, an incomplete queue is reflected on the archive tape, and records logically past the error are not returned to the system for reuse. BOTBOT is, however, advanced, thus leaving a chain of records unavailable but no longer part of any queue chain. This condition will be discovered in the file-check phase of on-line dumps. A REGENERATE would be required to return this space to the system.

7. After the archive tape is created, blocks of dumped records are returned to the available block mask. A printer file is written, reflecting file usage statistics as follows:

DATACOM QUEUE INPUT		
	FILE SIZE	INUSE
BEFORE DUMP	XXXXXXXXXXXX SEGMENTS	XXXXXXXXXXXX SEGMENTS
AFTER DUMP	XXXXXXXXXXXX SEGMENTS	XXXXXXXXXXXX SEGMENTS
		XXXXXXXXXXXX SEGMENTS
		RETURNED

DATACOM QUEUE OUTPUT		
	FILE SIZE	INUSE
BEFORE DUMP	XXXXXXXXXXXX SEGMENTS	XXXXXXXXXXXX SEGMENTS
AFTER DUMP	XXXXXXXXXXXX SEGMENTS	XXXXXXXXXXXX SEGMENTS
		XXXXXXXXXXXX SEGMENTS
		RETURNED

8. The last phase of each on-line dump is a file-check phase. A "snapshot" of the file usage tables and queue bottoms and tops is taken. Then each chain is traced through to insure that all blocks marked as unavailable are in fact in some chain. If blocks are found that belong to no chain, a DISPLAY is made and the records are not returned to the system.

MESSAGE: ON-LINE DUMP COMPLETED SEE PRINTER FOR FILE USAGE STATISTICS

In general, Datacom dumps should be taken shortly after data base dumps are completed. In this way, a short rebuild is required from any given on-line data base dump to the next, where GEMCOS archival recovery will begin.



# SECTION 10

## AUXILIARY PROGRAMS

This section describes auxiliary programs which may be used for message recall and administrative message switching.

### Message Recall

In the <global section> of the Transaction Control Language (TCL), the user is given the option of defining a Message Recall program to recall any message inserted into a queue on the current day, by its message number or by time-of-day. Only one program need be declared for all systems. The message key used within any system must be defined at the system level using the <recall message key statement> syntax. Only one master copy of the program is dynamically processed by the Message Control System (MCS) for all systems. The queue depth and time limit parameters may be set to invoke dynamic volume control.

A standard Recall program, a Process program, is supplied with the GEMCOS system. The name of the Object file is GEMCOS/RECALL.

The syntax of a recall message, as expected by the standard program, is as follows:

```
<recall message> ::=
    [declared message key from TCL] <period>
    <recall primitive> <period>

<recall primitive> ::=
    <recall message list> /
    TIME <slash> <time recall list> /
    LAST <slash> <integer> <destination description>

<recall message list> ::=
    <recall message part> /
    <recall message list>, <recall message part>

<recall message part> ::=
    <recall source> <message number range>
    <destination description> <message type>

<recall source> ::=
    (<station number>) <slash> /
    <station mnemonic> <slash> /
    (<input queue number>) <slash> / <empty>
```

<message number range> ::=  
     <message number> <dash> <message number> /  
     <message number>

<destination description> ::=  
     <slash> TO (<station number>) /  
     <slash> TO <station mnemonic> / <empty>

<message type> ::=  
     <slash> <message type identifier> / <empty>

<message type identifier> ::=  
     INPUT / IN / I / 1 /  
     OUTPUT / OUT / O / 2 /  
     INPUTOUTPUT / IO / 3 /  
     INPUTQUEUE / IQ

<time recall list> ::=  
     <time recall message> /  
     <time recall list>, <time recall message>

<time recall message> ::=  
     <recall source> <time of day range> <date specification>  
     <destination description> <message type>

<time of day range> ::=  
     <time of day> <dash> <time of day> /  
     <time of day>

<date specification> ::= <slash> ON <date> / <empty>

<date> ::= <month> <slash> <day> <slash> <year>

<station number> ::= <integer>

<message number> ::= <integer>

<inputqueue number> ::= <integer>

<time of day> ::=  
     [4 digit time of day ranging from 0000 to 2400]

<month> ::= [integer ranging from 1 to 12]

<day> ::= [integer ranging from 1 to 31]

<year> ::= [2 digit integer specifying year]

If <recall source> is an Input queue number, a station number, or a station mnemonic, then the message-number range is a range of system message numbers. If <recall source> is empty, then the message-number range is a range of station message numbers for the input station.

If the <message type identifier> is INPUT, then only the specified input messages are recalled; if OUTPUT, then only the specified output messages are recalled; if INPUTOUTPUT, then both the input and the corresponding output are recalled. If INPUTQUEUE is specified, then the <recall source> is assumed to be an Input queue number, and input messages for that particular queue are recalled.

The recall message by time-of-day allows the user to specify a series of time ranges to indicate the messages to be recalled. The time required is the time as it appears in the title of the output message.

A <message type identifier> of INPUTQUEUE may not be specified for a time-of-day recall.

If a date is specified on a time recall, then archival recall is assumed, and the network's audit ledger is scanned to determine on which archival dump tape the required messages reside. When the tape is made available to the MCS, it is scanned, and messages satisfying the recall request are extracted.

If the <recall primitive> LAST is specified, the last n output messages received at the terminal are recalled, where n is the <integer> specified in the message.

All recall messages may include a <destination description> which causes the recall response to be sent to the specified station. When specified, any recalled output message is formatted as if it were going to its original destination, regardless of the destination of the recall message.

Examples:

IRC.1410.	% Recall the output message % associated with station % message number 1410 for % this station.
IRC.TTY2/1415.	% Recall the output message % associated with system message % number 1415 for station TTY2.
IRC.10-12.	% Recall output messages % associated with station % message numbers 10-12 % for this station.
IRC.TTY2/10-12.	% Recall output messages % associated with system % numbers 10-12 for % station TTY2.

IRC.17-25/IO.	% Recall the input and % associated output messages % associated with station % message numbers 17-25 for % this station.
IRC.25/IO	% Recall the input and % associated output message % associated with % station message number 25 for % this station.
IRC.TTY2/1751/IO.	% Recall the input and % associated output messages % associated with system message % number 1751 for station TTY2.
IRC.(1000)/145/INPUT.	% Recall the input message % associated with system message % number 145 for station number 1000.
IRC.(3)/10/IQ.	% Recall the input message % associated with system message % number 10 for input queue % number 3.
IRC.1452,1515.	% Recall the output messages % associated with station message % numbers 1452 and 1515 for % this station.
IRC.TTY2/1323,(1212)/789.	% Recall the output message % associated with system message % number 1323 for station TTY2 % and the output message associated % with system message number % 789 for station number 1212.
IRC.TIME/1204,1305-1400.	% Recall the output messages % stamped with time 1204 and % between times 1305 through % 1400 for this station.
IRC.TIME/TTY2/1100-1215, (1000)/1100-1215/TO/TO TTY2.	% Recall the output messages % stamped with time between % 1100-1215 for station TTY2 % and the input and output % messages stamped with time % 1100-1215 for this station % and send the response to % station TTY2.
IRC.1420/TO (1000).	% Recall the output message % associated with station % message number 1420 for this % station and send the response % to station number 1000.

IRC.TIME/1210-1300/ON 05/06/76.	% Recall the output messages stamped % with time between 1210-1300 on % May 6, 1976 for this station.
IRC.LAST/5.	% Recall the last 5 outputs % received at this station.
IRC.LAST/2/TO TTY1.	% Recall the last 2 outputs % received at this station and send % them to station TTY1.

## Station-To-Station Administrative Messages

In the <global section> of the TCL, the user is given the option of defining an administrative message-switching program to route messages from one station to another station, to a list of stations, or to an area. Only one program need be declared for all systems. The message key used within any system must be defined at the system level using the <administrative message key statement> syntax. Only one master copy of the program is dynamically processed by the MCS for all systems. The queuedepth and timelimit parameters may be set to invoke dynamic volume control.

A standard administrative message-switching program is supplied with the GEMCOS System. It is a Process program. The name of the object file is "GEMCOS/ADMIN".

The syntax of an administrative message, as expected by the standard program, is as follows:

```
<administrative message> ::=
    [declared message key from TCL].
    <originating station name> <blanks>
    <destination list> .[TEXT TO BE SENT]

<destination list> ::=
    <area name> / <destination station list>

<destination station list> ::=
    <station name> /
    <destination station list, <station name>

<blanks> ::= [any number of spaces].
```

Example:

```
ADMIN.TTY2 TTY3. HI THERE
ADMIN.TTY2 TTY3,TD42, TC55 . EOJ IN 5 MINUTE5
```

The program receives the message and checks either for a valid destination area or for a list of valid station names. A check is also made for stations which were marked down. These stations are flagged, and their names are saved to be sent later with an acknowledgement message to the source station.



Only one area may be specified as a destination, and it must be the only entry in the <destination list>.

A limit of 15 station names may be specified in the <destination station list>.

If any errors are detected while syntaxing the message, the scanning ceases, and an error message is sent to the source station. The following is a list of detectable errors:

1. No destination name separator.
2. Invalid destination name.
3. Area name must be the only entry in the destination list.
4. Area name not allowed in station list.
5. A maximum of 15 stations allowed in destination list.
6. No period found to delimit destination list.
7. No destination name found.

If no errors are detected, a copy of the message is sent out to each station in the destination list. If a station in the list was marked down, an attempt is made by the MCS to route the message to the station's alternate. After routing the message, an acknowledgement message is sent to the source station along with a list of the stations which were marked down. If an area was specified, this name is sent with the acknowledgement.

# SECTION 11

## PROGRAM GENERATION AND MAINTENANCE

This section describes the files contained on the release tape and the procedures necessary to compile the MCS, the Transaction Control Language (TCL) processor, and the various other programs necessary to operate in a GEMCOS on-line environment. In addition, the steps are provided to generate the demonstration system which is included on the release tape.

Additional information about each current release may be obtained by loading and printing the distribution letter which is contained on each release tape in the form of a printer backup file. The following control cards should be used to load and print the distribution letter:

```
?COPY GEMCOS/MCSDOC FROM <tape name>
?PB "GEMCOS/MCSDOC"
```

### Contents of Release Tape

Listed in tables 11-1 thru 11-3 are the files contained on the tape of the TOTAL version of GEMCOS. For the BASIC and ADVANCED versions, appropriate files are omitted from the tape based on the features contained in the version.

Table 11-1. GEMCOS TOTAL Version

#### Files Required for Normal Usage

File Name	Description
GEMCOS/MCS	GEMCOS object code (DCALGOL).
GEMCOS/UTILITY	TCL Processor object code. Also provides dump and log generation utilities (DCALGOL).
GEMCOS/EDITOR	Skeleton Editor program object code. Performs standard input formatting. (DCALGOL).
GEMCOS/RECALL	Standard application-level program which retrieves audited messages for retransmission upon request. Object code (DCALGOL).
GEMCOS/ADMIN	Standard application-level program for station to station(s) message switching object code (DCALGOL).
GEMCOS/RESTART/SYMBOL	Source code of standard application-level program which retrieves certain DMS II restart information for a DMS II environment. User patches in specific data base, restart data set, and restart item names. (COBOL).
GEMCOS/RESTART74/SYMBOL	A Restart program written in COBOL74
GEMCOS/ARCHREC	Archival Recovery initiator object code (ALGOL).

Table 11-2. GEMCOS TOTAL Version

Optional Files	
File Name	Description
GEMCOS/MCSDOC	Product distribution letter (PB).
GEMCOS/SAMPLE/TCL	Example of TCL input (data).
GEMCOS/SAMPLE/TP	Source code of sample Transaction Processor(s) (COBOL and BDMSCOBOL).
GEMCOS/SAMPLE/TP74	Source code of sample Transaction Processor (COBOL74).
GEMCOS/SAMPLE/NDL	Source code of sample NDL request sets which follow the GEMCOS interface conventions (data).
GEMCOS/SAMPLE/SERVICE	Source code file of sample service program (COBOL).
GEMCOS/FORMATLIB/TCL GEMCOS/FORMATLIB/PATCH GEMCOS/SAMPLE/FORMATTP GEMCOS/FORMATTP/SYMBOL GEMCOS/SAMPLE/ALGOLLIB GEMCOS/SAMPLE/COBOLLIB GEMCOS/FORMATLIBRARY	These files may be used to demonstrate the use of a format library. They are described in Section 6, Formatting and Paging.
GEMCOS/FILEXFER	Sample Transaction Processor which uses ROUTEHEADERS to transfer files from one system to another. It is described in section 13, Computer to Computer Communication.
GEMCOS/FILEXFER/SYMBOL	Source code of GEMCOS/FILEXFER.

Table 11-3. GEMCOS TOTAL Version

Maintenance Files	
File Name	Description
GEMCOS/SYMBOL	Common source code file of MCS and Utility.
GEMCOS/ARCHREC/SYMBOL	Source code of Archival Recovery program.
GEMCOS/EDITOR/SYMBOL	Source code of standard Editor program.
GEMCOS/ADMIN/SYMBOL	Source code of standard Administrative Message Switching program.
GEMCOS/RECALL/SYMBOL	Source code of standard Message Recall Program.

## Product Generation

The MCS is not "generative" in the usual sense of the word. Once a system code file is created, it should not be necessary (unless a Ports interface is required) to recompile the program should the data communications environment change. In certain instances, it may be necessary to run the TCL compiler to regenerate the tables used by the MCS; however, certain on-line changes to TCL are provided (refer to section 8).

If it should be necessary to compile the MCS or the TCL processor, the following user-defined compiler options should be used.

Code Options	Description
\$SET UTILITY	Emits TCL processor code
\$SET MONITOR	Emits on-line monitor code. To be used only if compiling the MCS.
\$SET PORTS	Emits code for Port interface in MCS and TCL processor.
\$SET SUBSYSTEMS	Emits code for Subsystems interface in MCS and TCL processor.
\$SET SUPPRESSINTERCOM	Discard intercom messages.
\$SET UIO	Should be set when the MCS is to be used on a UIO datacom system.

MCS List Options	Description
\$SET SEEHANDLER	Complete listing of the MCS.
\$SET SEEGETMESSAGE	List procedure Get Message.
\$SET SESENDMESSAGE	List procedure Send Message.
\$SET SEEPROCESSEVERYTHING	List procedure Processeverything.
\$SET SEEBIGBROTHER	List procedure Big Brother.
\$SET SEEINPUT	List procedure Input.
\$SET SEEOUTPUT	List procedure Output.
\$SET SEEANALYZER	List procedure Analyzer.

TCL Processor List Options	Description
\$SET SEEUTILITY	Complete listing of Utility.

In addition, all of the standard DCALGOL options may be used, as discussed in the following paragraphs.

The following are the control cards necessary to compile the MCS with the monitor included and get a complete stack listing with LINEINFO set:

```
?COMPILE GEMCOS/MCS DCALGOL LIBRARY
?ALGOL FILE TAPE (TITLE = GEMCOS/SYMBOL)
?DATA CARD
$SET MERGE LIST SINGLE STACK LINEINFO
$SET MONITOR SEEHANDLER
?END
```

The following are the control cards necessary to compile the TCL processor and get a complete stack listing with LINEINFO set:

```
?COMPILE GEMCOS/UTILITY DCALGOL LIBRARY
?ALGOL FILE TAPE (TITLE = GEMCOS/SYMBOL)
?DATA CARD
$SET MERGE LIST SINGLE STACK LINEINFO
$SET UTILITY SEEUTILITY
?END
```

#### NOTE

If a complete listing of the MCS or the TCL processor is required, the list option must be used along with SEEHANDLER or SEEUTILITY.

GEMCOS supplies a fully documented source file of a sample Restart program to be used with DMS II. This sample demonstrates the proper passes of control from the Restart program to the MCS, as well as the logic flow within the program. This sample source file may be patched by the user to replace the naming conventions of the restart data set used by GEMCOS with the user's data names. In addition, if the user is using more than one data base in the data communications environment, the additional data base invokes and retrievals must be patched into the source.

The following are the control cards necessary to compile the Restart program:

```
?COMPILE GEMCOS/RESTART/PROG WITH BDMSCOBOL LIBRARY
?COBOL FILE TAPE(TITLE = GEMCOS/RESTART/SYMBOL)
?COBOL FILE NEWTAPE(TITLE = GEMCOS/RESTART/NEWSYMBOL)
?DATA CARD
  $SET MERGE LIST SINGLE STACK NEW
(COBOL SOURCE CARD PATCH DECK)
?END
```

## Product Maintenance

From time to time, it may be necessary to issue interim patches to update the standard GEMCOS release. These patches are issued in a form to be used as input to SYSTEM/PATCH which is supplied on the system software tape.

The following control cards should be used to update GEMCOS/MCS:

```
?JOB MCS/UPDATE;
```

```
BEGIN
```

```
?DATA MCS/PATCH
```

```
$#
```

(Insert here all the compiler \$ option cards necessary to control the compile.)

(Insert here the DCALGOL source card patch deck.)

```
?RUN SYSTEM/PATCH[T]; VALUE = 0
```

```
FILE PATCH (TITLE = MCS/MERGEDINPUT);
```

```
FILE TAPE (TITLE = GEMCOS/SYMBOL);
```

```
FILE CARD (TITLE = MCS/PATCH);
```

```
WAIT(T);
```

```
IF VALUE(T) = 1 THEN
```

```
BEGIN
```

```
COMPILE GEMCOS/MCS DCALGOL LIBRARY;
```

```
COMPILER FILE TAPE (TITLE = GEMCOS/SYMBOL);
```

```
COMPILER FILE CARD (TITLE = MCS/MERGEDINPUT);
```

```
END
```

```
?END JOB
```

The control cards to update GEMCOS Utility are the same as for updating GEMCOS/MCS except that "COMPILE GEMCOS/UTILITY" replaces "COMPILE GEMCOS/MCS".

The following control cards should be used to update GEMCOS/EDITOR:

```
?JOB MCS/UPDATE;  
BEGIN  
?DATA MCS/PATCH  
$#
```

(Insert here all the compiler \$ option cards necessary to control the compile.)

(Insert here the DCALGOL source card patch deck.)

```
?RUN SYSTEM/PATCH[T]; VALUE = 0;  
FILE PATCH (TITLE = MCS/MERGEDINPUT);  
FILE TAPE (TITLE = GEMCOS/EDITOR/SYMBOL);  
FILE CARD (TITLE = MCS/PATCH);  
WAIT(T);  
IF VALUE(T) = 1 THEN  
BEGIN  
COMPILE GEMCOS/EDITOR DCALGOL LIBRARY;  
COMPILER FILE TAPE (TITLE = GEMCOS/EDITOR/SYMBOL);  
COMPILER FILE CARD (TITLE = MCS/MERGEDINPUT);  
END  
?END JOB
```

The control cards to update GEMCOS/ADMIN and GEMCOS/RECALL are the same as for updating GEMCOS/EDITOR, with each occurrence of "EDITOR" replaced by "ADMIN" or "RECALL," respectively.

The following control cards should be used to update GEMCOS/ARCHREC:

```
?JOB MCS/UPDATE;  
BEGIN  
?DATA MCS/PATCH  
$#
```

(Insert here all the compiler \$ option cards cards necessary to control the compile.)

(Insert here the DCALGOL source card patch deck.)

```
?RUN SYSTEM/PATCH[T]; VALUE = 0;  
FILE PATCH (TITLE) = MCS/MERGEDINPUT;  
FILE TAPE (TITLE = GEMCOS/ARCHREC/SYMBOL);  
FILE CARD (TITLE = MCS/PATCH);  
WAIT(T);  
IF VALUE(T) = 1 THEN  
BEGIN  
COMPILE GEMCOS/ARCHREC ALGOL LIBRARY;  
COMPILER FILE TAPE (TITLE = GEMCOS/ARCHREC/SYMBOL);  
COMPILER FILE CARD (TITLE = MCS/MERGEDINPUT);  
END  
?END JOB
```

## Usage and Examples

Included on the GEMCOS release tape are two files which can be used to demonstrate the system and test the correct installation of the product.

The first file is GEMCOS/SAMPLE/TCL, a data file which can be used as input to the TCL processor to describe a specific data communications environment. The sample TCL may be used as it is except for the station names which must be patched over with names which are valid for the specific site. The following control cards should be used to generate the four files used by the MCS to describe the data communications environment:

```
?RUN GEMCOS/UTILITY
?FILE TAPE (TITLE = GEMCOS/SAMPLE/TCL)
?FILE NEWTAPE (TITLE = GEMCOS/SAMPLE/NEWTCL)
?DATA CARD
$MERGE LIST SINGLE NEW
```

(Patch cards to insert local station names over the sample station names.)

```
?END
```

The sample data communications environment uses three Transaction Processors, whose source is included as GEMCOS/SAMPLE/TP, plus a Restart program, whose source is included as GEMCOS/RESTART/SYMBOL. The sample Transaction Processors (TPs) were designed to thoroughly test all facets of the MCS's operations. It is unlikely that a User program would have to do many of the MCS functions exercised in the sample program.

To compile the non-DMS TP, the following control cards should be used:

```
?COMPILE GEMCOS/SAMPLE/PROG COBOL LIBRARY
?COBOL FILE TAPE (TITLE = GEMCOS/SAMPLE/TP)
?DATA CARD
$SET MERGE LIST SINGLE STACK
?END
```

To generate the access Control module, type the following command at a supervisory console:

```
COPY GEMCOS/SAMPLE/PROG AS GEMCOS/SAMPLE/ACCESS
```

To compile the Service program required by the sample data communications environment, the following control cards should be used:

```
?COMPILE GEMCOS/SERVICE COBOL LIBRARY
?COBOL FILE TAPE (TITLE = GEMCOS/SAMPLE/SERVICE)
?DATA CARD
$SET MERGE LIST SINGLE STACK
?END
```



If it is desired to use the TP which accesses a DMS II data base, the data base must be compiled first. The data base definition which is assumed by the sample TP is given by the following card images along with the control cards necessary to generate the data base files:

```
?COMPILE GEMCOSDB WITH DASDL LIBRARY
?COMPILER FILE NEWTAPE (TITLE = GEMCOS/SAMPLE/DASDL)
?DATA CARD
$SET LIST SINGLE NEW SEQ 100 + 100
GEMCOS-DB DATA SET
(GEMCOS-VALUE NUMBER(8));
GEMCOS-DB-SET OF GEMCOS-DB
KEY IS GEMCOS-VALUE DUPLICATES;
CONTROL-DATA DATA SET
(LAST-VALUE REAL (S11);
LAST-VALUE-KEY NUMBER (8));
CONTROL-SET SET OF CONTROL-DATA
KEY IS LAST-VALUE-KEY;
RESTARTAREA RESTART DATA SET
GEMCOS-LITERAL ALPHA(6);
GEMCOS-INTERFACE GROUP
  (GEMCOS-DATA REAL (S11);
  GEMCOS-DBSN REAL (S11);
  GEMCOS-SSN REAL (S11)) OCCURS 3 TIMES);
RESTARTSET SET OF RESTARTAREA
KEY IS GEMCOS-LITERAL DUPLICATES;
?END
```

After the data base is generated, and with the file "DESCRIPTION/ GEMCOSDB" present, the Transaction Processor which accesses the data base may be compiled. The following control cards should be used:

```
?COMPILE GEMCOS/SAMPLE/DMS/PROG WITH BDMSCOBOL LIBRARY
?COBOL FILE TAPE (TITLE = GEMCOS/SAMPLE/TP)
?DATA CARD
  $SET MERGE LIST SINGLE STACK DMS
  $SET WAITFORAUDIT RECOVERY
?END
```

#### NOTE

Users of the BASIC and ADVANCED versions should not use the WAITFORAUDIT and RECOVERY options in the above compilation. To do so results in a parameter mismatch between the program and the MCS during program initialization.

If using the TOTAL version, the Restart program should be compiled using the following control cards:

```
?COMPILE GEMCOS/RESTART/PROG WITH BDMSCOBOL LIBRARY
?COBOL FILE TAPE (TITLE = GEMCOS/RESTART/SYMBOL)
?DATA CARD
  $SET MERGE LIST SINGLE STACK
?END
```

The network may now be initialized. The name of the MCS must be one which is declared in the NDL. Assuming that "GEMCOS/MCS" is a valid MCS name in the NDL, and assuming that the DCP has been initialized (type DCn at the supervisory console, where n is the number of the DCP), the following control cards may be used:

```
?RUN GEMCOS/MCS
?END
```

If files were label-equated in the TCL run, they must be similarly label-equated here also. The sample uses the standard Editor supplied on the release tape, so this must be loaded before any transactions are processed.

An initialization message should appear at a Monitor station. The MCS is then up and running. The first thing which must be done prior to the input of transactions is to enable the network. This is done by typing the following Network Control command at a Network Control station:

```
?ENABLE ALL
```

The full list of Network Control commands is contained in Section 8.

Following is a list of transactions which may be used with the sample package:

Transaction	Result
LOGON.A	To gain access from a SIGNON = TRUE station.
LOGOFF.	To sign off of a SIGNON = TRUE station.
ECHO0.<string>	Echoes the string.
ECHO1.<integer>#<alpha>*	Edits and echoes the data. The order of appearance of the integer and alpha strings may vary.
ECHO2.<integer>#<alpha>*	Same as ECHO1 except different editing.
MONITOR.	Turns monitor function on in sample TP.
SERVICE.	Causes all subsequent input to SAMPLE/TP to be sent to to GEMCOS/SAMPLE/SERVICE.
DMS01.UPDATE	Opens the data base in Update mode.
DMS01.INQUIRY	Opens the data base in Inquiry mode.
DMS01.TEMPORARY	Open the data base in Temporary mode.
DMS01.CLOSE	Closes the data base.
DMS06.nnnnnn	Creates a data base record with integer value nnnnnn inserted as data.
DMS07.	Lists values of all records in the data base.



## SECTION 12

# MCS/NDL INTERFACE

In order to provide maximum possible efficiency, a certain amount of communication via TOGs and TALLYs between the Message Control System (MCS) and the Data Communications Processor (DCP) is necessary. The MCS expects the DCP to abide by certain conventions in this interface. We therefore provide the NDL programmer with the following information:

1. A detailed description of how TOGs and TALLYs are used by GEMCOS.
2. A sample request/control set for a TC 500 terminal.

In a test environment, the CANDE/RJE ORIENTED NDL request sets supplied on the system software tape will suffice; however, because of the integral association between a transaction-oriented MCS (GEMCOS) and the application-oriented parts of the NDL, the NDL conventions, as specified here, must be followed. In particular, the NDL should specify `ENABLEINPUT = FALSE` for all stations.

Tables 12-1 and 12-2 provide information regarding the use of TOGs and TALLYs. The example beginning on page 12-4 contains a sample NDL line control and transmit request set, both of which are oriented to the TC 500 terminal.

Table 12-1. MCS to DCP Conventions

Request Set Area	Definition
TOG(0):	TRUE means station is master over system. (Station is another computer which is in charge.)
TOG(1):	TRUE means the station is a Network Control station (refer to Section 3).
TOG(2):	TRUE means send broadcast select. Used to save time when all stations on a line are to receive a message.
TOG(3):	TRUE means station does not queue output. An input must occur after each output.
TOG(4):	Not currently used.
TOG(5):	Not currently used.
TOG(6):	Not currently used.
TOG(7):	TRUE means MCS has marked this message as one whose result is to be ignored. The NDL program should not change this field.
TALLY(0):	Counter set by MCS to station's line width.

Table 12-2. DCP to MCS Conventions

<b>Request Set Area</b>	<b>Definition</b>
TOG(0):	Not currently used.
TOG(1):	Set to TRUE whenever poll or select is in progress. Sent to MCS via WRUFLAG before doing TERMINATE ERROR.
TOG(2):	TRUE means station had transmission error.
TOG(3):	TRUE means input message had no STX. (Used only with Model 37 Teletypes)
TOG(4):	Not currently used.
TOG(5):	TRUE means message ended with ETB. Used with BISYNC line discipline.
TOG(6):	TRUE in good results message means DCP did TERMINATE BLOCK.
TOG(7):	Same as TOG(7) in Table 12-1.

[illegible]

```

% NOTE
%
% P
% 1. O IS LITTLE P CHARACTER
% L
%
% T
% 2. R IS OPTION 1 TO 3 DIGIT TRANSMISSION NUMBER.
% #
%
% A A
% 3. D D
% 1 2 ARE THE STATION'S RECEIVE CHARACTERS
%
% 4. B IS A BLOCK CHECK CHARACTER FORMED BY AN EXCLUSIVE
% C OR OF ALL CHARACTERS AFTER THE (SOH) TO AND
% C INCLUDING THE (ETX).
%
%
%
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

```

% THIS REQUEST SHOULD HANDLE B9342, TC500 & TD700
% & TD800
ERROR[0] = % POLL RESPONSE, RECEIVE TEXT OR JUNK
TIMEOUT: 700, %
LOSSOF CARRIER: 200, %
STOPBIT: 200, %
PARITY: 200, %
BUFOVFL: 200, %
BREAK: 200. %
%
ERROR[1] = % RECEIVE EOT FROM ACK TO MSG
TIMEOUT: 180, %
LOSSOF CARRIER: 180, %
STOPBIT: 180, %
PARITY: 180, %
BUFOVFL: 180, %
BREAK: 180. %
%
10: % START OF IT ALL, SEND POLL
%
INITIATE TRANSMIT. % GO TO TRANSMIT STATE,
TRANSMIT EOT [BREAK: 700]. % POLL FORMAT E A A P E
TRANSMIT ADDRESS (RECEIVE) [BREAK: 700]. % O D D O N
TRANSMIT POLENQ [BREAK: 700]. % T 1 2 L Q
%
20: %RECEIVE MESSAGE, EOT OR JUNK
%
FINISH TRANSMIT. % EXIT TRANSMIT STATE
INITIATE RECEIVE. % ENTER RECEIVE STATE
%
RECEIVE CHARACTER [ERROR[0]]. % WAIT FOR POLL
RESPONSE

```

```

%
IF CHARACTER EQL SOH THEN % MESSAGE COMING IN
BEGIN %
%
LINE(TOG[0]) = TRUE. % STATION ACTIVITY ON LINE
TOG [1] = FALSE. % POLL DONE, IN RECEIVE TEXT
% STATE
INITIALIZE BCC. % CLEAR BLOCK CHEC CHAR
% ACCUMULATOR
RECEIVE ADDRESS (RECEIVE) (ERROR[0], ADDR:200).
%
40: % FOR SKIPPING TRANSMISSION NUMBER
RECEIVE (ERROR[0], STX:50). %
GO TO 40. % UNTIL STX OR ERROR
50: % PREPARE FOR MESSAGE PIECE
GETSPACE (200). % NEED MESSAGE SPACE
INITIALIZE TEXT. % START STORING AT BEGINNING
% OF BUFFER
60: % RECEIVE LOOP
RECEIVE CHARACTER (ERROR[0], END: 100). %
%
70: %
STORE CHARACTER (ENDOFBUFFER: 90). %
%
GO TO 60. % RECEIVE MORE
%
90: % END OF BUFFER
%
TOG[6] = TRUE. % MULTIPLE RECORDS
TALLY[0] = CHARACTER. % SAVE CHARACTER NOT STORED
TALLY[1] = BCC. % SAVE BLOCK CHECK CHARACTER
TALLY[2] = RETRY. % SAVE OFF THRU TERMINATE
%
TERMINATE BLOCK. % GIVE TO MCS
%
CHARACTER = TALLY[0]. % RESTORE CHARACTER NOT STORED
BCC = TALLY[1]. % RESTORE BLOCK CHECK CHARACTER
RETRY = TALLY[2]. % RESTORE RETRY
PAUSE. %
GETSPACE (200). % NEED ANOTHER MESSAGE SPACE
INITIALIZE TEXT. % START AT BEGINNING OF SPACE
GO TO 70. % ENTER RECEIVE LOOP AT STORE
%
100: % END OF MESSAGE
%
RECEIVE BCC (ERROR[0], BCCERR: 200). % RECEIVE
% & CHECK

```



```

%
TERMINATE LOGICALACK. % GIVE TO MCS, WAIT FOR ACK
%
INITIATE TRANSMIT. % ENTER TRANSMIT STATE
TRANSMIT ACK [BREAK:NULL]. % ACK MESSAGE
FINISH TRANSMIT. % EXIT TRANSMIT STATE
INITIATE RECEIVE. % ENTER RECEIVE STATE
RECEIVE CHARACTER [ERROR[1]]. % WAIT FOR EOT
180: % WE WILL NOT CHECK EOT, OR RESPOND TO ERROR
% INDICATIONS
%
TERMINATE NORMAL. %
%
END ELSE % SOH FIRST CHAR
IF CHARACTER EQL EOT THEN % NO MESSAGE COMING
BEGIN %
INITIALIZE RETRY. %
195: %
TERMINATE NOINPUT. % NO INPUT FROM THIS STATION
%
END ELSE
BEGIN % JUNK RECEIVED, WAIT TILL LINE IDLE
200: % EAT ANYTHING LOOP
RECEIVE [ERROR[0], END: 700]. %
GO TO 200. %
END. %
700: % RECEIVING ERROR HANDLING
IF RETRY GTR 1 THEN % RETRY NOT EXHAUSTED
BEGIN %
DELAY (100 MILLI). %
RETRY = RETRY + 1. %
IF TOG[1] THEN % ERROR DURING POLL
BEGIN %
LINE (TOG[0]) = TRUE. % INDICATE LINE ACTIVITY
% FOR RETRY
TERMINATE NOINPUT. % RETRY NEXT POLL TIME
END. %
% ERROR DURING RECEIVE OF MESSAGE
IF TOG[6] THEN % MCS HAS PART OF MESSAGE
BEGIN % TELL HIM TO JUNK IT
710: TOG[0] = TRUE. % JUNK MSG INDICATOR
STORE TOG[0]. % PUT IN MSG
TALLY[2] = RETRY. % SAVE RETRY THRU TERMINATE
%
TERMINATE BLOCK. % GIVE TO MCS

```

```

%
RETRY = TALLY[2]. % RESTORE RETRY
TOG[6] = FALSE. % RESET MULTIPLE RECORDS
END. %
% NAK MESSAGE
INITIATE TRANSMIT. % ENTER TRANSMIT STATE
TRANSMIT NAK (BREAK=NULL). %
% AFTER SEVERAL NAK RETRYS, TRY REPOLLING RETRY
IF RETRY LEQ 3 THEN % TRY REPOLLING
BEGIN %
    TOG[1] = TRUE. %
    FINISH TRANSMIT. % EXIT TRANSMIT STATE
    GO TO 10. %
END. %
GO TO 20. % DO FINISH TRANSMIT & WAIT FOR MESSAGE
END. % RETRY GTR 1
%
% NO HOPE FOR STATION, TERMINATE ERROR TIME
750: %
760: WRUFLAG = TOG[1]. % FOR HANDLER ERROR MSG
%
TERMINATE ERROR. % STATION DOWN
%
% END OF REQUEST GENERALPOLLREQUEST
CONTROL POLL: %
%
LINE (TALLY[0]) = LINE (TALLY[0]) + 1. %
LINE (QUEUED) = TRUE. %
%
0: %
%
%
IF STATION GTR 7 THEN %
BEGIN %
%
%
%
PAUSE. %
%
STATION = STATION - 1. %
IF STATION (VALID) THEN %
    IF STATION (READY) THEN %
        BEGIN %
            LINE (QUEUED) = TRUE %
            IF STATION (QUEUED) THEN %
                INITIATE REQUEST. %
            IF STATION (ENABLED) THEN %
                BEGIN %
                    IF LINE (TOG[0]) THEN % INPUT OR
                                                                % OUTPUT ACTIVITY
                    BEGIN % POLL REST OF STATIONS ON LINE
                        GO TO 20. %
                    END. %
                    IF STATION (TALLY) GTR LINE (TALLY[1])

```

```

                                THEN %
        BEGIN %
            STATION(TALLY) = LINE(TALLY[1]).
            GO TO 0.
        END. %
20:    IF NOSPACE THEN % NO ENABLE INPUT SPACE
        BEGIN %
            STATION (TALLY) = 0. %
        END ELSE %
        BEGIN %
            STATION (TALLY) = STATION
                                (FREQUENCY). %
            TOG[1] = TRUE. %
            INITIATE ENABLEINPUT. %
        END. %
    END. %
    END. %
    GO TO 0. %
    END. %
    STATION = MAXSTATIONS. %
    IF LINE (QUEUED) THEN %
        BEGIN %
            LINE (QUEUED) = FALSE. %
            IF LINE (TALLY[0]) = 0 THEN %
                BEGIN %
                    LINE (TALLY[0]) = 1. %
                    LINE (BUSY) = FALSE. %
                    DELAY (1 SEC). %
                    LINE (BUSY) = TRUE. %
                END ELSE %
                BEGIN %
                    IF LINE (TOG[0]) THEN % INPUT OR
                                                % ACTIVITY LAST
                                                % TIME THRU
                    LINE (TALLY[1]) = 250 ELSE % FOR FAST
                                                REPOLL
                    LINE (TALLY[1]) = LINE (TALLY[0]). %
                    LINE (TOG[0]) = FALSE. %
                    LINE (TALLY[0]) = 0. %
                END. %
            GO TO 10. %
        END.
        %
        IDLE. %
        %

```

```

REQUEST SELECTTC500:  %
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
%
%                               SELECT                               %
%                               |                                   %
%          COMPUTER            |               TERMINAL           %
%          EAASE              |                                   %
% (SELECT):->ODDEN  -----+----->                             %
%          | T12LQ          |                                   %
%          |                |                                   %
% (RETRY SELECT) <-----+----- (NO VALID RESPONSE)           %
%                               |                                   %
%                               |      N                          %
% (RETRY MESSAGE <-----+-----A (NOT READY TO RECEIVE)      %
%          LATER)        |      K                              %
%                               |                                   %
% (COMPLETE MESSAGE      |      A                              %
% IS LESS THEN <-----+-----C (READY TO RECEIVE)           %
% BUFFER SIZE)          |      K                              %
%                               |                                   %
%          SAATS           EB |                                   %
% :--> ODDRT [TEXT] TC  --+----->                             %
%          H12#X           XC |                                   %
%          |               |                                   %
%          | (COMPLETE MESSAGE IS |                                   %
%          | GREATER THAN BUFFER |                                   %
%          | SIZE, FIRST OR      |                                   %
%          | INTERMEDIATE BLOCK) |                                   %
%          |               |                                   %
%          | SAATS           DEB |                                   %
% :--> ODDRT [TEXT] CTC  +----->                             %
%          H12#X           1XC |                                   %
%          |               |                                   %
%          | (COMPLETE MESSAGE IS |                                   %
%          | GREATER THAN BUFFER |                                   %
%          | SIZE, LAST BLOCK)   |                                   %
%          |               |                                   %
%          | SAATS           EB |                                   %
% :--> ODDRT [TEXT] TC  --+----->                             %
%          H12#X           XC |                                   %
%          |               |                                   %
%          |               |      N                              %
% :--> (RETRY MESSAGE) <--+-----A (RECEIVE ERROR)           %
%          |               |      K                              %
%          |               |                                   %
%          |               |      A                              %
% (CONTINUE WITH <-----+-----C (RECEIVE OK)               %
% LINE)              |      K                              %
%                               |                                   %

```

```

% NOTE
%
% S
% 1. E IS THE LITTLE Q CHARACTER.
% L
%
% A A
% 2. D D IS THE STATION TRANSMIT ADDRESS CHARACTERS.
% 1 2
% B
% 3. C IS A BLOCK CHECK CHARACTER FORMED BY AN EXCLUSIVE
% C OR OF ALL CHARACTERS AFTER THE (SOH) TO
% AND INCLUDING THE (ETX).
%
% D
% 4. C IS A DEVICE CONTROL CHARACTER USED TO INDICATE
% 1 MORE BLOCKS TO COMPLETE MESSAGE.
%
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
%
% ERROR[0] = %
%   TIMEOUT: 200. %
%   LOSSOF CARRIER: 200. %
%   STOPBIT: 200. %
%   PARITY: 200. %
%   BUFOVFL: 200. %
%   BREAK: 200. %
%
% LINE (TOG[0]) = TRUE. % INDICATE LINE ACTIVITY
%
5: % START OFF WITH A SELECT
% INITIATE TRANSMIT. % GO TO TRANSMIT STATE
% SEND SELECT.
% TOG[1] = TRUE. % INDICATE SELECT IN PROGRESS
% TRANSMIT EOT [BREAK:200]. % SELECT FORMAT: E A A S E
% TRANSMIT ADDRESS (TRANSMIT) [BREAK:200]. % O D D E N
% TRANSMIT SELENQ [BREAK:200]. % T 1 2 L Q
% FINISH TRANSMIT. % EXIT TRANSMIT MODE
% INITIATE RECEIVE. % ENTER RECEIVE MODE
%
% RECEIVE CHARACTER [ERROR[0]]: % WAIT FOR ACK, NAK OR
% % SOMETHING
%
% IF CHARACTER EQL ACK THEN % STATION READY TO RECEIVE
% % MESSAGE
% BEGIN %
%
% TOG[1] = FALSE. % SELECT DONE
% TOG[6] = FALSE. %

```

```

10:      % REPEAT MESSAGE ENTRANCE
      INITIATE TRANSMIT.  % ENTER TRANSMIT MODE
      TRANSMIT SOH [BREAK:200].  %
      INITIALIZE BCC.  % CLEAR BLOCK CHEC CHAR
                        % ACUMULATOR
      TRANSMIT ADDRESS (TRANSMIT) [BREAK:200].  %
      % TRANSMISSION NUMBER NOT USED, BUT NEED ONE
      TRANSMIT TRAN [BREAK:200].  %
      TRANSMIT STX [BREAK:200].  %

20:      %
      INITIALIZE TEST.  %
      % SINCE MESSAGE WILL NOT BE ALTERED, REGULAR SEND
      % WILL DO
      TRANSMIT TEXT [BREAK:200].  %
      % END OF MESSAGE OR BLOCK
      IF BLOCK THEN %
      BEGIN %
          TOG[6] = TRUE.  % BLOCKED OUTPUT
          TALLY[1] = BCC.  % SAVE THRU TERMINATE
          TALLY[2] = RETRY.  % SAVE THRU TERMINATE
          %
          TERMINATE BLOCK.  %
          %
          BCC = TALLY[1].  % RESTORE
          RETRY = TALLY[2].  % RESTORE
          GO TO 20.  %
      END.  %
      % MESSAGE DONE
      IF PAGE THEN % FIRST BUFFER OR PAGE OF MESSAGE
      TRANSMIT DC1 [BREAK:200].  % TELL HIM WITH DC1
                                % BEFORE ETX
      TRANSMIT ETX [BREAK:200].  %
      TRANSMIT BCC [BREAK:200].  %
      FINISH TRANSMIT.  % EXIT TRANSMIT MODE
      INITIATE RECEIVE.  % ENTER RECEIVE MODE
      % WAIT FOR ACK, NAK OR SOMETHING ELSE
      RECEIVE CHARACTER [ERROR[0]].  %
      IF CHARACTER EQL ACK THEN % MESSAGE MADE IT
                                % TO TERMINAL
      BEGIN %

```

```

110:      % TERMINATE NORMAL EXIT
          %
          TERMINATE NORMAL. %
          %
      END ELSE %
      IF CHARACTER EQL NAK THEN % ERROR
      BEGIN %
          NAKFLAG = TRUE. % FOR POSSIBLE ERROR
                          % MESSAGE
          END. %
          % NAK OR SOMETHING ELSE IS REPEATED THEN
          % RETRY SELECT
          GO TO 200. %
      END ELSE % ACK RESPONSE
      IF CHARACTER EQL NAK THEN % NOT IN RECEIVE
      BEGIN %
          % RETRY LATER
          NAKONSELECT = TRUE. %
          GO TO 110. % TERMINATE NORMAL EXIT
      END. %

200:      % MANY ASSORTED ERRORS LEAD TO HERE
      IF RETRY GTR 1 THEN % RETRY NOT EXHAUSTED
      BEGIN %
          RETRY = RETRY - 1. %
          IF TOG[1] THEN % ERROR DURING SELECT
          BEGIN %
              % EXIT REQUEST AND RETRY LATER
              TERMINATE. %
          END. %
          % RESPONSE ERROR
          IF TOG[6] THEN % BLOCKED OUTPUT, TELL MCS
                          % TO RESEND
          BEGIN %
              NAKFLAG = TRUE. %
              %
              TERMINATE NORMAL. %
              %
          END. %
          IF RETRY GTR 2 THEN % RESEND MESSAGE
          GO TO 10. %
          GO TO 5. % RETRY AT SELECT
      END. %

900:      % TERMINATE ERROR TIME
      WRUFLAG = TOG[1]. % FOR HANDLER ERROR MSG
      TERMINATE ERROR. %
      % END OF SELECTTC500 REQUEST

```

# SECTION 13

## COMPUTER-TO-COMPUTER COMMUNICATION

GEMCOS provides two mechanisms for computer-to-computer communication. Both rely on GEMCOS adding a routeheader to the front of messages sent to a station. GEMCOS expects that messages sent to this station will be routed to another GEMCOS on a different system, although the user may provide his own program on that system which follows one of the two GEMCOS routeheader conventions described below.

The routing header is passed to a Process program in the controlwords array (refer to Appendix C). A user program must make a special request of the MCS if it wants to see the routing header. The appropriate passes of control are presented in Appendix A.

### GEMCOS ROUTEHEADER CONVENTIONS

#### Variable Field Routeheader

This type of routeheader is provided for a station which has ROUTEHEADER = TRUE specified in the TCL. The maximum length of this routeheader is 42 characters. This includes a 1-character signal at the head plus a 1-or 2-character terminator. The terminator indicates how the message associated with the routing header is to be treated by the MCS. If the terminator is a period, the MCS assumes the message is an original message and routes it to the appropriate application program for processing. If the terminator is a carriage-return-line feed, the MCS assumes the message is a return message (probably a response to an original message) to be delivered to a station.

The MCS associates with each input from a ROUTEHEADER = FALSE station a default routing header which is used in the event the message subsequently has to be switched to another computer or message concentrator. The form of this default header is as follows:

- 1 byte : Signal character (&)
- 6 bytes : STA
- 2 bytes : CODE
  - 0 Normal station input (STA = TCL station number)
  - 1 Normal station input (TP-to-TP message) (STA = TCL station number)
  - 8 Object/Batch job input (STA = Object serial number)
  - 9 Object/Batch job input (TP-to-TP message) (STA = object serial number)
- 5 bytes : Input message number
- 1 byte : Period

Before releasing a message to a ROUTEHEADER = TRUE station, the MCS attaches the routing header associated with the input message. An application program may override the use of this saved routing header by setting appropriate controls in COMMON. If MSG-OVRRTEHDR (COMMON[4].[42:1]) is set to value 1, the program indicates that the routing header to be used is contained at the head of the message text (always terminated by a period).



By default, the form of the routing header is determined by the destination of the message. If the destination is the station which originated the input, the message is formed as a return message for which the header is terminated by a CR-LF. If the destination is not the originating station, the message is formed as an original message, and the header is terminated by a period. An application program may override the default message form by setting appropriate controls in COMMON. If MSG-RETRTEHDR (COMMON[4].[43:1]) is set to value 1, the message is formed as a return message regardless of the originating station.

## Fixed Field Routeheader

This type of Routeheader is provided for a station which has TYPE = ROUTEHEADER, specified in the TCL. The same format of Routeheader is available for GEMCOS on the medium, small and CMS systems.

The format of this routeheader is as follows:

ROUTEHEADER type	2 bytes
Access key	6 bytes
Origination type	2 bytes
Return ID	14 bytes
Message number	10 bytes
Format ID	12 bytes
Error code	2 bytes

## Field Definitions

ROUTEHEADER Type – Used to indicate the type of routeheader sent or received. Possible values are:

- 0 = Standard, originating routeheader
- 1 = Standard, returning routeheader
- 2 = Recovery, originating
- 3 = Recovery reply
- 4 = Suspension
- 5 = Resumption

Access Key – This field will contain an accesskey to be validated by the receiving station.

Origination Type – This field indicates how the return routeheader is to be handled. The meaning of the value contained in the return ID field is evaluated according to this field. Possible values are:

- 0 = Return to station indicated in return ID field
- 1 = Return to inputqueue indicated in return ID field

Return ID – contains either a station or input queue number

Message number – This field will be used by a future release for recovery.

Format ID – This field will contain a message ID to be used in formatting.

Error Code – This field will be used by one host to inform the other of an error condition.  
Possible values are:

- 0 – No error
- 1 – Message key missing
- 2 – Program/station not available
- 3 – User has no access to program
- 4 – Invalid accesskey
- 5 – Illegal routeheader type
- 6 – Routeheader table full, message not processed
- 7 – Length of message less than 48 bytes
- 8 – Invalid input format request
- 9 – Destination is not a program
- 10 – Format error

### Return Messages to Program

If a program sends originating messages to a ROUTEHEADER station, i.e. to a station other than the one that the input message came from, GEMCOS will add an originating ROUTEHEADER (type = 0) with the origination type set to 0 (Return to originating station). If the TP wishes the reply to be directed back to itself, it may pass control to GEMCOS with MSG-DESTTYPE (COMMON[4].[39:81]) set to 11.

### Access Control

Any message sent to a ROUTEHEADER station will have the HOSTACCESSKEY inserted in the access key field of the ROUTEHEADER. If no HOSTACCESSKEY has been declared, blanks will be used.

If a ROUTEHEADER station has been declared to be SIGNON, the access key in the ROUTEHEADER will be checked for each message received to ensure that it is a valid access code for this station. If it is, the message will be checked to see if it is allowed with the access code. If either check fails an error will be sent to the originating host.

### Suspension & Resumption

When GEMCOS receives a suspension ROUTEHEADER from a station, it will set a suspension flag for that station. While this flag is set, no originating messages will be routed to that station, but return messages will still be sent. A message from a station to a suspended ROUTEHEADER will be rejected with a "busy" response and a message from a program will be rejected with error 139. The suspension flag is reset when a resumption ROUTEHEADER is received. Large systems GEMCOS does not issue suspensions or resumptions, although one could be issued from a TP by building it in the COMMONAREA and setting the MSG-OVRRTTEHDR field (COMMON[4].42:1) to 1.

## Formatting

### Formatting at the originating host

1. Messages sent from a station to a ROUTEHEADER station will not be formatted.
2. Messages originating from a TP going to a ROUTEHEADER station will have the MSGID supplied by the TP placed in the ROUTEHEADER and no formatting will be done.
3. Return messages received from a ROUTEHEADER station going to a station will be formatted using the MSGID in the ROUTEHEADER.
4. Return messages received from a ROUTEHEADER station going to a program will not be formatted.

### Formatting at the receiving host

1. Messages sent to a station (by message key) will not be formatted.
2. Messages sent to a program will have normal input formatting by message key. When a reply is received from a TP with a MSGID supplied, GEMCOS will attempt to do normal output formatting. If no format is found corresponding to the MSGID and the format field in the ROUTEHEADER is blank, the MSGID will be placed in the ROUTEHEADER. This can then be used by the originating host for formatting the reply back to the originating station.

### Example:

In this example it is assumed that there are two large systems for ease of TCL description, although in principle, this example is applicable to any two computer systems. Each computer will run a GEMCOS MCS. Those TCL statements which deal with ROUTEHEADERS are presented here for both computer 1 and computer 2.

#### TCL for Computer 1

```
BEGIN
  ACCESSCONTROL:
    ACCESSKEY HOST2 = ALL.
    ACCESSKEY USER1 = ALL.
%
  PROGRAM PROGRAM1 (1) USER:
    INPUTQUEUE I1(1):
      MKE = PGM1.
%
  STATION STATION1:
    SIGNON                = TRUE.
    VALIDACCESSKEYS       = USER1.
    MKE                    = STN1.
%
  STATION HOST2
    TYPE                   = ROUTEHEADER.
    SIGNON                  = TRUE.
    VALIDACCESSKEYS         = HOST2.
    HOSTACCESSKEY           = HOST1.
    MKE                     = PGM2, STN2.
END;
```

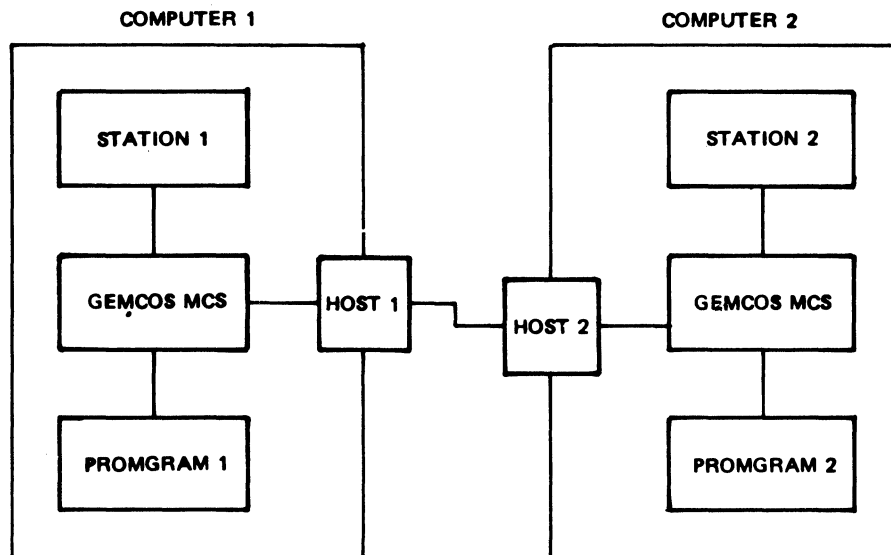
# TCL for Computer 2

```

BEGIN
  ACCESSCONTROL:
    ACCESSKEY HOST1 = ALL.
    ACCESSKEY USER2 = ALL.
%
  PROGRAM PROGRAM2 (2) USER:
    INPUTQUEUE 12(2):
      MKE = PGM2.
%
  STATION STATION2:
    SIGNON          = TRUE.
    VALIDACCESSKEYS = USER2.
    MKE             = STN2.
%
  STATION HOST1:
    TYPE            = ROUTEHEADER.
    SIGNON          = TRUE.
    VALIDACCESSKEYS = HOST1.
    HOSTACCESSKEY   = HOST2.
    MKE             = PGM1, STN1.
END;

```

A graphic representation of the above TCL specifications on each computer is shown in the following diagram:



#### ON HOST1:

1. If STATION1 enters a message that has a message key of STN2, the message is routed to computer 2 through the ROUTEHEADER station and delivered to STATION2.
2. Any message entered that contains the message key PGM2 is routed, in similar fashion, to PROGRAM2 on computer 2. The output from PROGRAM2 is automatically routed back to the originator on computer 1.

#### ON HOST2:

1. If STATION2 enters a message that has a message key STN1, the message is routed to computer 1 through the ROUTEHEADER station and delivered to STATION1.
2. Any message entered that contains the message key PGM1 is routed, in similar fashion, to PROGRAM1 on computer 1. The output from PROGRAM1 is automatically routed back to the originator on computer 2.

## FILE TRANSFER PROGRAM

A file transfer program was written as a test for ROUTEHEADERS and for internal use. It is included on the release tape as an example for users of the kind of thing that can be done using ROUTEHEADERS. A compatible program is also available on CMS, small and medium systems.

A user can enter a command from an MCS-controlled station which will specify the name of a file to be transferred to or from another computer. The file transfer program (MCSFILXFER) will attempt to transfer the file and inform the user when the transfer is complete or has failed. The program will transfer only one file at a time. Blocking and record size will be maintained during a file transfer. All data is transmitted in "transparent" mode, i.e., non-character data files can be transferred.

The following rules must be adhered to when defining the attributes of the file transfer program on the program section of the TCL:

1. The program must be declared as a USER program.
2. The COMMONSIZE statement must be set to a minimum value of  $((\text{RECORDSIZE} * 2) + 27)$  words, where RECORDSIZE is that of the file to be transferred.
3. The inputqueue number must be the same as the program number.

Included on the GEMCOS release tape is a fully functional file transfer object program called MCSFILXFER. The user need only declare this program in TCL.

## Operation

There are five commands which a user can enter to the file transfer program. The commands are COPY, ABORT, WHAT, NEWTC and CHGTC. The syntax and semantics for these commands follows:

## COPY Command

### Syntax:

```
<copy command> ::=  
    COPY <file name 1> <as clause>  
    <destination clause> <origination clause>  
  
<as clause> ::= AS <file name 2> / <empty>  
  
<destination clause> ::=  
    <direction> <destination trancode>  
  
<origination clause> ::=  
    USING <origination trancode> / <empty>  
  
<direction> ::=  
    TO / FROM  
  
<destination trancode> ::= <message key>  
  
<origination trancode> ::= message key>  
  
<file name> ::=  
    <quote> <identifier> <file name part>  
    <pack name part> <quote>  
  
<file name part> ::= <slash> <identifier> / <empty>  
  
<pack name part> ::=  
    ON <identifier> / <empty>
```

### Semantics:

This command is used to initiate the file transfer mechanism. <file name 1> is the name of the file on the system which is sending the file. <file name 2> is the name of the file on the system which is receiving the file. If <file name 2> is not specified, it will default to <file name 1>. Whenever a file name is specified for a large system, the above syntax for <file name> must be adhered to.

The keyword "TO" specifies that a disk file is to be sent to the remote computer from the local computer. The keyword "FROM" specifies that a disk file is to be sent from the remote computer to the local computer.

The <destination trancode> must be a valid trancode (message key) for the remote computer's file transfer program running under a GEMCOS MCS. The user must know this trancode. This trancode and any other possible destination trancodes that the user may wish to use for a particular computer (ROUTEHEADER station) must be declared for that ROUTEHEADER station on the local computer.

The <origination trancode> specifies the trancode which the remote file transfer program will use to communicate with the local file transfer program. If not specified, it defaults to "FTPLS." On a large system, "FT4800" on a medium system, "FT1800" on a small system and "FTCMS" on CMS machines. This trancode and any other possible origination trancodes that the user may wish to use, including the default value of FTPLS., must be declared for the file transfer program on the local computer.

## ABORT Command

Syntax:

<abort command> ::= ABORT

Semantics:

The abort command instructs the file transfer program to discontinue the current file transfer, if one is in progress. When the file transfer program receives this command, it instructs the file transfer program on the remote host to discontinue the current transfer, regardless of who initiated the request. This command can be entered at any time.

## WHAT Command

Syntax:

<what command> ::= WHAT

Semantics:

This command returns the current status of the file transfer program to the requestor. The possible responses are available and busy. If busy is returned, the file transfer program will not accept a copy command.

## NEWTC Command

Syntax:

<newtc command> ::=  
 NEWTC <destination trancode> ,  
 <destination description> ,  
 <system type> ,  
 <routeheader station number>

<destination trancode> ::= <quote> <trancode> <quote>

<destination description> ::= <quote> <literal> <quote>

<system type> ::= <integer>

<routeheader station number> ::= <integer>

#### Semantics:

The newtc command allows users to make run time changes to the destination trancode information contained in the "DEFINETCS" routine of the file transfer program. This information is used by the file transfer program to identify the remote system and its associated routeheader station. The information entered through this command is temporary will not be saved at program end of job.

The <trancode> is identical to the <destination trancode> described in the "COPY" command description.

The <destination description> is a 1 – 12 character literal which is used in the response to a "WHAT" command and in the "COPY COMPLETED" and "COPY ABORTED" messages. If left blank, the destination description will default to the destination trancode.

The <system type> is a 1-digit integer which identifies the type of the remote system. Valid values for this field are: 1 for CMS, 2 for small systems, 3 for medium systems, and 4 for large system.

The <routeheader station number> must be the same as that declared for the routeheader station in the TCL.

## CHGTC Command

#### Syntax:

<chgtc command> ::=  
    CHGTC <destination trancode>,  
        <destination description>,  
        <system type>,  
        <routeheader station number>

<destination trancode> ::= <quote> <trancode> <quote>

<destination description> ::= <quote> <literal> <quote>

<system type> ::= <integer>

<routeheader station number> ::= <integer>



## Semantics:

The CHGTC command allows users to change the destination description, system type, and routeheader station number for an existing trancode. As with the "NEWTC" command, these changes are temporary. The "CHGTC" command will not be allowed if the status of the file transfer program is "BUSY". The fields for this command are identical to those described for the NEWTC command.

## Example:

Assume that a file transfer is to occur between large systems. The computer that will initiate the file transfer request will be called HOSTA and the other computer will be called HOSTB. Further assume that the following statements exist in the TCL specifications for each GEMCOS MCS on each system.

### GEMCOS TCL on HOSTA

```
PROGRAM FTA (1) USER:           % FILE TRANSFER PROGRAM ON HOSTA
  MKE = FT68A, COPY, ABORT, WHAT.
STATION RHA (12):                % ROUTEHEADER STATION TO HOSTB
  MKE = FT68B.
  TYPE = ROUTEHEADER.
STATION TD830A (11):             % STATION WHERE COMMANDS WILL
                                % BE ENTERED
```

### GEMCOS TCL on HOSTB

```
PROGRAM FTB (2) USER:           % FILE TRANSFER PROGRAM ON
                                % HOSTB
  MKE = FT68B, COPY, ABORT, WHAT.
STATION RHB (21):                % ROUTEHEADER STATION TO HOSTA
  MKE = FT68A.
  TYPE = ROUTEHEADER.
```

A user at station TD830A on HOSTA (under control of a GEMCOS MCS) wishes to transfer the disk file MONDAY/NEWS to HOSTB with a file name of MONDAY/NEWS on diskpack BACKUP. The following dialog accomplishes the requested file transfer.

```
(FROM TD830A) : COPY "MONDAY/NEWS" AS "MONDAY/A/NEWS ON
               BACKUP" TO FT68A [12] USING FT68A
(FROM FTA)    : COPY INITIATED
(FROM TD830A) : WHAT
(FROM FTA)    : FILE TRANSFER PROGRAM BUSY TRANSFERRING
               FILE: MONDAY/NEWS TO SYSTEM: FT68B
(FROM FTA)    : COPY OF FILE: MONDAY/NEWS TO SYSTEM: FT68B
```

When file transfer occurs between two large systems, only one (or neither) file transfer program can use the default origination message key of FTPLS. In this example, the default message key was not used.

# APPENDIX A

## COMMON AREA LAYOUT

Following is the layout of the control portion of the Common area. The field in parentheses following the name is the location of this field for a USER program.

MSG-CONTROL (COMMON[0]) is used to indicate who is or should be in control of the Common area. The User program's control-bit value is 0. The Editor and Service programs' control-bit values are specified in the TCL, and the MCS has a control-bit value of 1.

COMMON[1] is used to define the pass of control to the receiver of the Common area. Since only passes of control to and from the MCS are system-defined, only these values can be defined.

COMMON[1].[47:21] is not defined.

MSG-PASSER (COMMON[1].[26:3]) identifies who is passing control to the MCS. This is valid only when the user passes the Common area to the MCS. Following is a list of the values and their meanings.

Value	Meaning
0	User program
1	Service program
2	(Not defined)
3	Editor program
4-7	(Not defined)

MSG-ACTION (COMMON[1].[23:24]) contains information about the Common passed to the MCS or to a program.

The following values are valid only when the user passes the Common area to the MCS.

Value	Meaning
0	First or intermediate part of a message is in the Common area to be sent to the destination. The length and position of the message is determined by COMMON[7]. The destination is determined by COMMON [4]. The MCS saves text and accumulates it with all subsequent value 0 passes of control text until a value 2 or value 3 pass of control, at which time the entire accumulated message is released.
2	End of current message to destination, but the program has more messages to send. COMMON[7] is checked for a message part. The message is sent to the destination, and control of the Common area is returned to the requesting program.
3	End of current message to destination. The program is finished with the input message. COMMON[7] is checked for a message part. The message, if any, is sent to the destination. The current input message is marked as processed.

Value	Meaning
10	Program is requesting that the next piece of input message be placed in the Common area. As much of the message as will fit in the Common area is returned, starting in COMMON[23]. The length of the message part is placed in COMMON[7].
11	Program is requesting the station name of the originator of the current input message. The name is placed in the Common area starting at COMMON[23] for COMMON[7] characters.
12	Program is requesting that the name in the text portion of the Common area whose length is in COMMON[7] be validated as a known station or area name. What type of name it should be is determined by a field in COMMON[4]. If the name is valid, its number is returned in COMMON[7].
13	Program is requesting that the current routing header be returned. It is returned in COMMON[23] for COMMON[7] characters.
14	Access Control program is requesting a log-on to be verified. COMMON[23] contains a valid access control key for COMMON[7] characters.
15	Program is requesting that any message being accumulated for a destination be aborted and not sent. Control of the Common area is returned to the requesting program.
16	Access Control program is requesting a log-off to be verified.
17	To be used by a Restart program only. Indicates that the program is passing back to the MCS a valid sorted list of recovery information. The length of the sorted list is in COMMON[7].
18	To be used by a Restart program only. Indicates an error was detected by the Restart program
19	To be used by a Restart program only. Indicates to the MCS that the program has deleted all restart information from the restart data set
20	Program is reporting that it received an ABORT exception on a DMS II BEGIN or END TRANSACTION.
21	Access Control program is requesting that all stations in the "system" be logged off.

The following values are valid only when the MCS passes the Common area to a program.

Value	Meaning
0	Passed to a Restart program to tell it not to delete the restart information from the restart data set.
5	Last pass of control to the MCS was in error. COMMON[7] contains the error indicator.
6	Last pass of control to the MCS was completed without error. COMMON[7] contains the result indicator.
7	A new message for an Editor program.
16	Part of the input message is in the Common area starting at COMMON[23] for COMMON[7] characters. There is still more to the message.

Value	Meaning
17	Passed to a Restart program to tell it to delete the restart information from the restart data set.
19	Last part of input message is in the Common area, or the originating station name is in the Common area starting at COMMON[23] for COMMON[7] characters.

COMMON[1] is set to value 8 whenever the MCS is processing a Common area.

COMMON[2].[47:24] is not defined.

MSG-STA-NO (COMMON[2].[23:24]) contains the message originator reference number. If the originator is a station, the reference number is the station number; if the originator is an Object job, the reference number is the job number; and if the originator is a Batch job, it is the File Relative Station Number (FRSN).

MSG-MODE (COMMON[3].[47:1]) identifies the mode of the originating station when the message was received. Value 0 means the station was in data mode, and value 1 indicates that the station was in practice mode.

COMMON[3].[46:1] value 1 indicates that the program is a test program.

COMMON[3].[45:1] is not defined.

MSG-ASSIGN (COMMON[3].[44:1]) indicates whether a station was assigned to a program. Value 0 means the station was not assigned; value 1 indicates that it was.

MSG-ORIOBJORSTA (COMMON[3].[43:1]) identifies the origin of the message. Value 0 means the message originated from a station; value 1 indicates that the message originated from an Object job using the originating station as a reference station for the Object file.

MSG-MSGTYPE (COMMON[3].[42:3]) indicates the type of input message. Following is a list of this field's values and their meanings.

Value	Meaning
0	A normal station message.
1	The message was generated by a program, but the response is sent to the station at which the original input message was entered.
2	The message is a system-defined input message.
3	The message was generated by a program, and the response is sent to the originating program.
7	This is a FLUSHRECOVERY indicator to the program. The systems have just had a failure, Input and Output queues have been flushed, and the program is being informed (refer to Sections 3 and 5).

MSG-SSN (COMMON[3].[39:22]) contains the system input message number, the unique message number assigned to the message at input.

MSG-ORINUM (COMMON [3].[17:18]) contains the number of the originating station.

COMMON[4] provides message destination control information for the MCS. The program uses this word to determine the destination of a response to an input message. It is set to 0 when an input message is passed to a program. A value 0 implies that all subsequent responses are sent to the originator of the message.

If a message is passed by the application program in a number of pieces (COMMON[1] = 0) for accumulation by the MCS before being released to the destination, this word determines the destination of the message only at the first pass of control.

COMMON[4].[47:4] is not defined.

MSG-RETRTEHDR (COMMON[4].[43:1]) should contain value 1 if the response is to be formatted as a return message. This affects only destination stations that use routing headers and which are different from the originator. (A detailed discussion pertaining to routing headers is presented in Section 13.)

MSG-OVRRTEHDR (COMMON[4].[42:1]) should be value 1 if the message contains a routing header that is to override the one saved by the system. The routing header is at the front of the message terminated by a period. A response message may start immediately after the period. This is valid only when the first piece of the message is sent and is included in the message length as specified in COMMON[7]. (A detailed discussion pertaining to routing headers is presented in Section 3.)

MSG-TITLE (COMMON[4].[41:2]) affects the date-time title placed at the front of all responses. The values are as follows:

Value	Meaning
0	The building is to be determined by the destination of the message.
1	No title is to be built.
2	The title destination part is to contain the multiple destination identification.

MSG-DESTTYPE (COMMON[4].[39:8]) specifies the destination of the message. The values and their meanings are:

Value	Meaning
0	Destination is a station. The station's number is in COMMON[4].[19:20].
1	Destination is an area. The area number is in COMMON[4].[19:20].
2	Destination is a Transaction Processing program. The response is directed back to the terminal from which the original transaction started.
3	Destination is the System Monitors defined in the Definition deck.
4	Destination is the System Network Controllers defined in the Definition deck.
7	Destination is a Transaction Processing program. The primary response is directed back to the originating program (not to a terminal).

Value	Meaning
8	The name to be validated is a station name. This is only to be used with a value 12 in COMMON[1]. [23:24]. If the name is valid, the station number is returned in COMMON[7].
9	The name to be validated is an area name. This is only to be used with a value 12 in COMMON[1]. [23:24]. If the name is valid, the station number is returned in COMMON[7].
10	Destination is a station. The station name is in COMMON[11], [12], and [13], terminated by 4"FF".
11	Destination is a station. If it is a ROUTEHEADER = TRUE station and not the originator of the message, then the return field in the ROUTEHEADER will be set to this program's number and any return messages will be directed back to this program.

MSG-CRLF (COMMON[4].[31:8]) is used to control the carriage return and line feed characters sent at the end of a message to an unbuffered terminal. Following are the values for this word.

Value	Meaning
0	Send the carriage return and two line feeds at the end of the message
1	Send carriage return at the end of the message
2	Send two line feeds at the end of the message
3	Send nothing at the end of the message

MSG-CONV-END (COMMON[4].[23:1]) indicates End-of-Conversation. The next input from the destination station produces nulls in the program's Conversational area.

MSG-CONV-DEALLOC (COMMON[4].[22:1]) tells the MCS to deallocate the station's Conversational storage area. This field is examined only if COMMON[4].[23:1]=1.

MSG-CONV-CORESTORE (COMMON[4].[21:1]) tells the MCS to store the program's Conversational area in the station's in-core Conversational storage area.

MSG-CONV-DISKSTORE (COMMON[4].[20:1]) directs the MCS to store the program's Conversational area on the station's audit trail.

MSG-DEST (COMMON[4].[19:20]) identifies the destination of a message. If the value is 0 and the destination is a station-by-station number, the destination is the originator of the input message. If the destination is the System Monitors or the System Network Controllers, this field is irrelevant. If the destination is a station-by-station number or an area, this field is the appropriate number. If the destination is a program and the value is 0, the message is routed according to the message key contained in the message; if the destination is a program and the value is non-zero, this field is the TCL-defined program number of the destination program.

COMMON[5] contains the message time variable, which is the date and time of the input message.

MSG-DATE (COMMON[5].[47:24]) contains the date of the input message. Value 120175 would be December 1, 1975. The value of the date is in binary form.

MSG-TIME (COMMON[5].[23:24]) contains the time of the input message in sixtieths of a second for a USER program or seconds for a PORT program.

COMMON[6] contains information about the input message dealing with the transaction processing program.

COMMON[6].[47:1] should be value 1 if the program is disabled. This is true only during recovery.

COMMON[6].[46:1] should be value 1 if the Input queue is in a rerun state. Rerun is controlled by a CHANGE INPUTQUEUE System Control message.

MSG-RETRY (COMMON[6].[45:3]) contains the transaction retry-counter, which is the number of times this transaction was submitted to the TP and caused it to terminate abnormally. This field is incremented by 1 whenever an input transaction causes a TP to terminate abnormally.

MSG-RECOVERY (COMMON[6].[42:2]) indicates the recovery status of the system. Following is a list of its values.

Value	Meaning
0	The system is not in recovery mode.
1	The system is in recovery mode caused by a TP abort.
2	The system is performing an archival recovery.
3	The system is in recovery mode caused by a Halt/Load or an abnormal termination of the MCS.

MSG-DISPLAY (COMMON[6].[40:1],) when value 1, indicates that the message is a display type message. It indicates that an UPDATE format is to be used as a DISPLAY ONLY format. The MCS does not reset this bit to its normal value of zero.

COMMON[6].[39:4] is not defined.

MSG-EDIBIT (COMMON[6].[35:8]) contains the control-bit value of the Editor program associated with the User program and is used to pass control to the Editor program. The value is undefined if the User program has no associated Editor program.

COMMON[6].[27:8] (not needed for a PORT program) contains the control-bit value of the Service program associated with the User program and is used to pass control to the Service program. The value is undefined if the User program has no associated Service program.

COMMON[6].[19:8] (not needed for a PORT program) contains the USER program event number and is used by the Service and Editor programs to pass control to the User program.

COMMON[6].[11:4] is not defined.

MSG-PROG-NUM (COMMON[6].[7:8]) contains the program number associated with the input message. The association is by message key or by the assigned state of the originator.

COMMON[7] contains general message length information, MCS result indicators, or the area or station number in response to a validation request. Message length information is different for messages to a User program than for messages from a User program. MCS result indicators are passed to programs after a bad pass of control.

COMMON[7] – Result Indicators. This field contains result information passed to a program from the MCS. This is dependent on the original pass to the MCS. Result indicators with a value greater than 127 are passed with COMMON[1] equal to 5. Following are the values for this word.

Value	Meaning
0	Request completed with no errors.
64	Request completed with no errors, but the destination station is disabled with no alternate station to handle the output.
128	Message destination error, station or area name to be converted is unknown.
129	Send to Area denied because there are currently no stations in the area.
130	Indicates a data error. The program is trying to send a nonqueued output message of a length greater than 3000 characters.
131	Indicates an originator error. The destination is valid, but the originator may not send to it (probably a cross-system transmission).
132	Indicates that the program has requested a nonqueued output message sent to a nonoperative (NOP) station.
133	Access control violation. This is returned to a non-Access Control Program requesting a log-on or log-off.
135	Message part not within Common area.
136	Undefined pass of control.
137	Received in response to a TP-to-TP and WAIT request. Destination TP inoperative or aborted while processing the transaction.
138	Received in response to a TP-to-TP and WAIT request. The TP-to-TP response was larger than the Common area of the originating TP.
140	Message destination error, a non-queued output message may not be sent to multiple stations.
143-148	Access Control Violation – returned only to an Access Control Program.
143	Undefined access key.
144	Access key not allowed at station.
145	Too many users.
146	User already logged on.
147	Someone else already logged on.
148	Station is assigned (log-off denied).



COMMON[7] – Area or Station Number. If a request was made to validate an area or station name and the name was valid, the number associated with the name is returned.

COMMON[7] – Message Length from MCS. This field contains the character length of the input message to a User program and a format error indicator.

MSG-FMTERR (COMMON[7].[47:1]) contains the format-error bit. If an error occurs during input formatting, the Editor turns this bit on to indicate the situation to the TP. Since the message contents are suspect, the TP should not process the message normally.

MSG-LENGTHIN (COMMON[7].[46:47]) contains the character length of an input message to a User program and is the number of characters starting at COMMON[23], or after the Conversation area, if one exists.

COMMON[7] – Message Length from User Program. This field indicates the length of the message part to be sent to the destination. It is passed from the program to the MCS.

MSG-QBYPASS (COMMON[7].[46:1],) at value 1, indicates that the message is to bypass the disk queueing, i.e., it is sent directly to the destination station without going to disk and without being accumulated into a completed message. The length must be 1 greater than the true length of the message. Because the message is not audited, it will be lost if the station is not ready to receive it on the first try. Output formatting will not be done. Output accumulation is not possible.

COMMON[7].[45:6] is undefined.

MSG-OFFSET (COMMON[7].[39:20]) contains the character offset from COMMON[23] where the message starts.

MSG-LENGTHOUT (COMMON[7].[19:20]) indicates the character length of the message.

MSG-STABITS (COMMON[8]) contains the station bit variable, i.e., the station bits of the originator of the message. This is where a User-type Access Control module indicates the value to which to change the station bits during a successful log-on or log-off.

MSG-MESSAGEID (COMMON[9]) contains the output message-ID, indicating which format, if any, is to be applied to this output message. COMMON[9] contains spaces placed by the MCS on an initial pass of control of an input message. The message-ID should be left-justified with trailing blanks.

MSG-RESTART-DATA (COMMON[10]) contains recovery information. The data in COMMON[10] should be placed in the restart data set if the program is involved with data base recovery (refer to Section 5).

COMMON[10].[47:8] identifies the Input queue number of the queue in which this message resides (as defined in the TCL).

COMMON[10].[39:8] contains the index into the MCS program table.

COMMON[10].[31:24] contains the input disk address of this input message.

MSG-REL-IQU (COMMON[10].[7:8]) identifies the relative Input queue which this message was in.

MSG-STA-NAMEWORD (COMMON[11] – COMMON[13]), upon the initial pass of control to a User program, contain the name of the station originating the message terminated by 4"FF". If the station name is 18 characters or longer, only the first 18 characters are stored. If the entire name is required, a special pass of control exists to acquire it in the text portion of COMMON, i.e., MSG-ACTION (COMMON[1].[23:24]) = 11.

If the user wishes to route output to a station based upon the station name, the name is placed in these words terminated by 4"FF". If the station name is 18 bytes long, the 4"FF" is not required.

MSG-MODULE and MSG-FUNCTION (COMMON[14] – COMMON[15]) contain the module index and function index associated with the message key.

MSG-USER-ID (COMMON[16].[47:8]) contains the 1-character ID used to differentiate between users of a multi-user log-on station. (Details on Access Control are presented in Section 7.)

MSG-ITEM-COUNT (COMMON[16].[39:16]) indicates the item count associated with the message key.

COMMON[16].[23:22] is undefined.

MSG-DUPAUDIT (COMMON[16].[1:1]) is set to value 1 if the current input transaction was audited in duplicate. All output related to this input is also audited in duplicate. If the TP does not want certain output messages audited twice, it may designate this by resetting this bit to value 0. If subsequent output is to be audited in duplicate, the bit must be set back to 1.

If duplicate audit files exist, a program can cause any output to be audited in duplicate by setting this bit to 1, even if the corresponding input was not audited in duplicate.

MSG-WAITAUDIT (COMMON[16].[0:1]) is set to value 1 if the program is going to wait for output audit. (This is the WAITFORAUDIT bit discussed in Section 5.)

MSG-DBSN (COMMON[17]) contains the Data Base Sequence Number (refer to Section 5).

COMMON[18].[47:47] is not defined.

MSG-TRANSTATE (COMMON[18].[0:1]) is set to value 1 if the program is in transaction state, and value 0 if the program is not in transaction state. It is set and reset by the User program. (This is the TRANSACTIONSTATEBIT discussed in Section 5.)

MSG-MTCHAR (COMMON[19].[47:8]) is set by the MCS to the character received in the header from a TYPE = MT600 station and may be set by the program to a character to be sent in the header to a MT600 terminal. If the program sets it to blank no header is sent.

MSG-MTDEL (COMMON[19].[39:1]) is set to 1 by a program to inform the MCS that a trailer must be appended to this message for sending this message to a MT600 terminal.

COMMON[19].[38:39] is not defined.

MSG-CONV-SIZE (COMMON[20].[47:12]) is the program's Conversation size in words.

COMMON[20].[35:36] is undefined.

MSG-USERID1 (COMMON[21]) if STATION INDIVIDUALID = TRUE, contains the first word of the user identification for the station that originated the input.

MSG-USERID2 (COMMON[22]) if STATION INDIVIDUALID = TRUE, contains the second word of the user identification for the station that originated the input.

COMMON[23] – COMMON[N] is a general message area used to pass information between the MCS, User, Editor, and Service programs. An input message passed to a User program without using an Editor starts at word 23, or after the Conversation area (if one exists) and, depending on the length, could fill up the entire Common area.

# APPENDIX B

## DATACOM SYSTEM DISK FILES

This appendix describes the disk files used by the Data Communications System. Since a great amount of data is necessary to operate a Data Communications System, this data is saved on disk for convenience and quick access.

The Data Communications System obtains from disk pertinent data communications network information and keeps on disk a copy of all input and output messages, as well as real-time information. The pertinent data communications network information is kept in a control file. A copy of all input and output messages are kept in the queue files.

### DATA COMMUNICATIONS UTILITY FILES

The data communications Utility program maintains certain files to accomplish its tasks. They may be label-equated to suit the user's needs.

The following is a list of the internal names, external names, and file descriptions of all files generated and/or referenced by the Utility program:

Internal Name	External Name	File Description
CARD	CARD	Card reader, 80-character/record source images
LINE	LINE	Line printer, 132-character/record source and <list> output listing
TAPE	DATACOM/DEFINITION/DECK	Disk, 14-word records, blocked 30 library source images
NEWTAPE	DATACOM/DEFINITION/NEWDECK	Disk, 14-word records blocked 30 <new> library source images
CQUS	DATACOM/QUEUE/CONTROL	Disk, 30-word records, unblocked Control Queue file
IQUS	DATACOM/QUEUE/INPUT	Disk, default 30-word records, unblocked Input Queue file
OQUS	DATACOM/QUEUE/OUTPUT	Disk, default 60-word records, unblocked Output Queue file

Internal Name	External Name	File Description
FMTFILE	DATAACOM/QUEUE/FORMATS	Disk, 30-word records, unblocked Format file
CQUSO	DATAACOM/OLDQUEUE/CONTROL	Disk, 30-word records, unblocked old Control Queue file
IQUSO	DATAACOM/OLDQUEUE/INPUT	Disk, default 30-word records, unblocked old Input Queue file
OQUSO	DATAACOM/OLDQUEUE/OUTPUT	Disk, default 60-word records, unblocked old Output Queue file
FMTFILEO	DATAACOM/OLDQUEUE/FORMATS	Disk, 30-word records, unblocked old Format file
IQUSDUP <i>IQUSODUP</i>	DATAACOM/QUEUE/DUPLICATE/INPUT	Disk, default 30-word records, unblocked duplicate Input Queue file
OQUSDUP <i>OQUSODUP</i>	DATAACOM/QUEUE/DUPLICATE/OUTPUT	Disk, default 60-word records, unblocked duplicate Output Queue file
CQUSDUP <i>CQUSODUP</i>	DATAACOM/QUEUE/DUPLICATE/CONTROL	Disk, 30-word records, unblocked duplicate Control Queue file
LOGIQUSO	DATAACOM/QUEUE/INPUTLOG	Disk, 18-word records, blocked 10 Input Message log
LOGOQUSO	DATAACOM/QUEUE/OUTPUTLOG	Disk, 18-word records, blocked Output Message log
ADTLEGFILE	DATAACOM/LEDGER	Disk, 10-word records, blocked 3 ledger of archival dumps
ADTTAPEFILE	ARCHIVEFORSYSnnn/ DT <6-digit date> TM <4-digit time>, where nnn = system number.	Tape, 60-word records, blocked 5 archival audit

## SYSTEM CONTROL FILE

The Data Communications System Control file contains pertinent data communications network and real-time information. The pertinent information is read-only, table information. It remains constant day after day until the data communications network is redefined. The real-time information is written out at a certain rate and reflects the system condition at a certain time.

The Control file is called DATACOM/QUEUE/CONTROL. The record size is 30 words (180 bytes), and the file contains enough records to handle the required storage. Words 0-28 (bytes 0-173) are data words, and word 29 (bytes 174-179) is a link word.

The following tables describe the different records of the Control file.

Record zero – System Variables:

Word	Name	Description
0	COMMAX	Number of programs allowed to run at once
1	SCRMAX	Length of LSN Scramble table
2	LINMAX	Length of Line tables
3	STAMAX	Length of Station tables
4	PROMAX	Length of Program tables
5	IQUMAX	Length of Input Queue tables
6	AREMAX	Length of Area tables
7	MKEMAX	Length of Message Key table
8	STATIONNAMESMAX	Length of Station Names table
9	PROGRAMNAMESMAX	Length of Program Names table
10	INPUTQUEUEENAMESMAX	Length of Input Queue Names table
11	AREANAMESMAX	Length of Area Names table
12	SYSTEMNAMESMAX	Length of System Names table
13	SYSTEMMONITORLINK	Station links to System Monitors
14	SYSTEMSPOLINK	Station links to System Network Control stations
15	ALTMAX	Length of Alternate Station Table
16	SECURITYMAX	Length of Security Item table
17	TASKMAX	Total number of tasks needed for system
18	MKEMESSAGEMAX	Length of Message Key (MKE) message table
19	STATIONBITSMAX	Length of Station Bits table
20	INTERNALMCSMAX	Number of line analyzers defined
21	CONTROLPAG	Maximum number of stations allowed in PAGER
22	HALTLOADWORD	Indicates last shut-down status
23	CONTROLFLAGS	Flags for optional functions in control process
24	SECURITYITEMMAXBIT	Number of security items defined

Word	Name	Description
25	OLDTABLELINKS	Links to prior control information
26	CCLOCK[1]	File start date
27	STARTTIME	File start time
28	CQUSLAST	Link word to next variable record

Record one – Table Link Addresses. For each word in the following list, WORD[35:18] is the starting key for the table, and WORD[17:18] is the last key.

Word	Table Name
0	LINIOD
1	STAIAD
2	STATIONS
3	PROGRAMS
4	INPUTQUEUES
5	AREAS
6	MESSAGEKEYS
7	STATIONNAMES
8	PROGRAMNAMES
9	INPUTQUEUEENAMES
10	AREANAMES
11	SYSTEMNAMES
12	ALTERNATE
13	SECURITYITEMS
14	MKEMESSAGE
15	DCCODEDATE & DCCODETIME
16	STADCPATTRIBUTES
17	STATIONBITS
18	BATCHIOBACKUP
19	USERCODE
20	STASECURITYMASK
21	MKETABLE
22	FAMILYTITLE
23	USERSECURITYMASK
24	MESSAGEID
25	FORMATS
26	DEVICENAMES
27	SYSTEMS
28	BACKUP
29	SYSTEMMKES

Record Two – System Variables:

Word	Name	Description
0	SYSTEMWRD	Contains system data
1	SYSMAX	Length of Systems table
2	STALOGMAX	Length of Station Log-on table
3	USERMAX	Length of User Code table
4	FAMMAX	Length of User Family table
5	MKEUSERMAX	Length of User Security Mask table
6	MSGIDMAX	Length of Message-ID table
7	MKETBLMAX	Length of MKE table
8	MSGFMTMAX	Number of message keys and output-IDs related to formats
9	DEVICEMAX	Length of Device Names table
10	USERSTAMAX	Length of Station Security Mask table
11	SYSMKEMAX	Length of System MKES table
12	FMTCOL	Maximum column length of Formats table
13	SEQCONTROL	Sequence Control number
14	USEDOLD	Old file has been used to create new files
15	PRODUCTION-MODE	Production or test file
16	MAXNEWMKES	Maximum MKES that may be added before TCL regenerate required. Default is 5.
17	MAXNEWACCESS-KEYS	Maximum access keys that may be added before TCL regenerate required. Default is 5.
18	MAXNEWFORMATS	Maximum formats that may be added before TCL regenerate required. Default is 5.
19	BATCHMAX	Number of Batch jobs allowed. Default is 0.
20	FCMMAX	Number of stations with FORMS-COMPOSE specified
21	SWITCHSTAMAX	Size of table holding indices for those stations allowed to switch systems
22	STAAREMAX	Rowsize of station areas table
23	MSGIDTBLMAX	Maximum number of message-IDs that may use output routing
24	MSGIDBLROW	Rowsize of message-ID table
25	STACNVMAX	Maximum stations with in-core conversations
26	BASEMASKREC	First record of available block masks
27	QUEUEBLOCKSIZE	Size of blocks allocated in queue files
28	PORMAX	Number of Port programs defined



### Record Three – Links to Tables

Word	Name	Description
0	FORMSCOMPARRAY	Points to start of FORMSCOMPOSE dictionary in Control file
1	SYSTEMSECYREF	Points to start of system security tables in Control file
2	STATIONAREAS	Points to start of STATIONAREAS dictionary in control file
3	MSGIDTABLE	Points to start of message-ID dictionary in Control file
4	HOST NAMES	Points to start of Host dictionary in Control file
5	SUBSYSTEM NAMES	Points to start of Subsystem dictionary in Control file
6	ENVIRONMENT NAMES	Points to start of Environment dictionary in Control file
7-29	---	Undefined

### Master Backup Record

#### Backup Record 0:

Word	Name	Description
0	SYSDBSN	Last DBSN used
1	CCLOCK[0]	Last backup date
2	XCLOCK	Last backup time
3	QUESEG[0]	Highest IQUS record assigned
4	QUESEG[1]	Highest OQUS record assigned
5	QUESEG[2]	Highest CQUS record assigned
6	MSGSDAT	Input messages since midnight
7	SYSTEMFLAGS	System operating flags
8	BATCHMSGSDAY	Batch messages since midnight
9	MASKRECSALLOCED	Number of records in available block mask link
10	I	Link
11-29	---	Not defined

## Minor Backup Records

Backup Records 1 to (IQUMAX DIV 7) + 1:

Word	Description
0-27	Queue status, 4 words per Input queue
28-29	Not defined

Backup Records 2 + (IQUMAX DIV 7) to (STAMAX DIV 4) + 3:

Word	Description
0-24	Output queue status, 6 words/station
25-29	Not defined

## SYSTEM DISK QUEUE FILES

The data communications system disk queue files contain a copy of all input and output messages. Each message is placed on disk before it is processed. A message is placed into the Input Queue file as it is obtained from the data communications network and into the Output Queue file before it is sent to the data communications network. This queueing of messages allows complete message integrity.

Each message in the queue is composed of one or more records. The last record of the message, called the base record, is linked to the next base record. Intermediate records of a message, if any, are linked to the base record and to each other by a link. Each record of a message contains a message record number so that record order within a message can be maintained. Other message-dependent data is also saved in each record of a message.

The Data Communications Disk queue files are composed of two files. All input messages are placed in a file called "DATACOM/QUEUE/INPUT". All output messages are placed in a file called "DATACOM/QUEUE/OUTPUT". The default record size for input is thirty (30) words, 180 bytes; for output, sixty (60) words, 360 bytes. Each file holds a total of 16384000 records, 2000 areas of 8192 records. Random accessing is used on both files.

### INPUT QUEUE FILE

The Input Queue file contains a copy of all input messages. They are placed in the queue prior to processing and remain there for logging and recovery purposes after processing. The messages in the input queue are divided into subqueues, one for each queue defined in the TCL. These subqueues are linked together within themselves. When a new message is put in a subqueue, an unused record address is assigned from the Input Queue file.

A normal subqueue might be a queue of all messages which have no valid message type. The first message of a subqueue always starts at a reserved record address. The end (top) of a subqueue is obtained by linking from the starting point to the last valid message via the base record link.

Words 0-10 contain Control information, and the message text begins at word 11. Below is the layout of the first 11 words of each record in the Input Queue file. The data portion of the record would follow immediately.

Word	Bits	System Field Defines	Description
0	47:8	IQUTYE	Type of input message: Value 0 – Normal station input Value 1 – Generated by a program; IQUSTA indicates originating station Value 7 – Restart message for FLUSH RECOVERY IQUS Value 8 – Object I/O input; IQUSTA is OBJECTSTA. Value 9 – Message generated by a program; original message generated by an object job. IQUSTA is OBJECTSTA Value 12 – Batch I/O input; IQUSTA is OBJECTSTA. Value 13 – Message generated by program; original message generated by a batch job. IQUSTA is the Object program's FRSN.
0	39:1	IQUBSE	Value 1, if and only if this is a base record
0	38:1	IQUFRT	Value 1, if and only if this is the first record of the message
0	37:2	---	Not defined
0	35:3	IQUPRS	Number of times input has been processed
0	32:1	IQUDEL	Value 1, if and only if the MTS input had a trailer deleted
0	31:8	IQUREC	Record number of a record within a message
0	23:24	IQUSTA	Station index of the originating station
1	47:8	IQUISCL	Program index of the TP for which the message is destined
1	39:8	IQUIQU	IQU index in which this message resides
1	31:8	IQUFRC	First character position of the message after the Control Words
1	23:24	IQUMSG	Input message number by day
2	47:1	IQUFMT	Value 1, if and only if this message requires a format
2	46:20	IQUCHR	Number of valid characters in the message
2	26:8	IQUFTC	First text character of the message after the message key
2	18:11	IQUMKC	First character position of the message key (if one exists), or else equal to IQUFTC
2	7:8	IQUFMC	First graphic character of the message. Normally equals IQUMKC unless station uses routing header; then points to routing header
3	47:8	IQUID	User identification character

<b>Word</b>	<b>Bits</b>	<b>System Field Defines</b>	<b>Description</b>
3	39:17	IQUDAY	MCS Date Stamp – MM DD YY format
3	22:23	IQUTIM	MCS Time stamp = TIME (1) when message record processed
4	47:24	IQULST	Valid only on a base record and only if IQUFRT is not equal to 1. Equal to IQUNXT of next-to-last record of message (where the base is the last record)
4	23:24	IQUSIL	The originating station backward input link
5	47:24	IQUFST	On a base record with IQUFRT not equal to 1, then link to the first record of the message; on the first record (IQUFRT = 1) then if object or batch input then the FRSN else the TCL number of the originating station.
5	23:24	IQUNXT	If a base record, link to next base record; otherwise link to next record of message. On next-to-last record (where the base is the last), equals IQULST of the base record
6	47:24	IQUSTP	Output file address where the primary output generated by this message will probably be audited
6	23:24	IQUFPT	Valid only if IQUFMT = 1. Pointer to the format description for message
7	47:16	IQUMKY	Message key index for this message
7	31:8	IQUTRM	Information about the terminal type
7	23:24	IQUCNX	Index to conversation in output audit trails
8	47:48	IQUSBT	Value of station bits (if any) for the station which sent the input
9	47:48	IQUII1	First word of the user identification for the station which sent the input
10	47:48	IQUII2	Second word of the user identification for the station which sent the input

#### OUTPUT QUEUE FILE

The Output Queue file contains a copy of all output messages. They are placed in the queue prior to release to the station and remain there for logging and recovery purposes. The messages in the Output queue are divided into subqueues, one for each station. Each record within a subqueue is linked to the base record, to the next base record of a multirecord message, and to the input that generated the output. In addition, all outputs associated with an input, for whatever station, are linked together.

Words 0-9 contain Control information, and the message text begins at word 10. Below is the layout of the first ten words of each record in the Output Queue file. The message text of the record would follow immediately.

Word	Bits	System Field Defines	Description																						
0	47:4	OQUTYE	Original input message information (see IQUTYE).																						
0	43:4	OQUTYP	Type of message: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Normal message</td></tr><tr><td>1</td><td>Program to program; originator is a station.</td></tr><tr><td>2</td><td>This message not sent. Message sent via Broadcast to some other station on line</td></tr><tr><td>3</td><td>This message sent to all stations on line via Broadcast to this station</td></tr><tr><td>4-5</td><td>Not defined</td></tr><tr><td>6</td><td>Response of a TP-to-TP message being returned to the originating TP</td></tr><tr><td>7</td><td>Object I/O Key MSG from PRO to STA</td></tr><tr><td>8</td><td>OBJECT I/O output</td></tr><tr><td>9</td><td>Program to program originator is an OBJECT/BATCH job</td></tr><tr><td>15</td><td>Object I/O Key MSG from STA to PRO</td></tr></table>	Value	Meaning	0	Normal message	1	Program to program; originator is a station.	2	This message not sent. Message sent via Broadcast to some other station on line	3	This message sent to all stations on line via Broadcast to this station	4-5	Not defined	6	Response of a TP-to-TP message being returned to the originating TP	7	Object I/O Key MSG from PRO to STA	8	OBJECT I/O output	9	Program to program originator is an OBJECT/BATCH job	15	Object I/O Key MSG from STA to PRO
Value	Meaning																								
0	Normal message																								
1	Program to program; originator is a station.																								
2	This message not sent. Message sent via Broadcast to some other station on line																								
3	This message sent to all stations on line via Broadcast to this station																								
4-5	Not defined																								
6	Response of a TP-to-TP message being returned to the originating TP																								
7	Object I/O Key MSG from PRO to STA																								
8	OBJECT I/O output																								
9	Program to program originator is an OBJECT/BATCH job																								
15	Object I/O Key MSG from STA to PRO																								
0	39:16	OQUMEC	Control for the end of the message																						
0	23:1	OQUDUP	True, if and only if output is dublicately audited																						
0	22:23	OQUESTA	Station index of the destination																						
1	47:24	OQUSER	File relative sequence number of the Object or Batch job or the TCL-defined station number																						
1	23:24	OQUMSG	Input message number																						
2	47:1	OQUBSE	Value 1, if and only if this is a base record																						
2	46:1	OQUFRT	Value 1, if and only if this is the first record of the message																						
2	45:2	-----	Not defined																						
2	43:20	OQUCHR	Number of valid characters in the message (including the title and routing header)																						
2	23:8	OQUFRC	The first character position of the message after the Control Words																						
2	15:8	OQUSCM	Starting character position of the text (after the title and routing header)																						
2	7:8	OQUSCT	Starting character position of the title. If there is no routing header, this equals zero.																						
3	47:1	OQUFRQ	Value 1, if and only if this is a forms request																						
3	46:1	OQUPRM	Value 1, if and only if this is a primary output																						
3	45:1	OQUORI	Value 1, if and only if the destination station was the originator of the associated input message																						

Word	Bits	System Field Defines	Description
3	44:1	OQUTLK	Value 1, if and only if this message was linked into the front of the output queue
3	43:1	OQUDSP	Value 1, if and only if this is a display type message
3	42:1	OQUHLD	Value 1, if and only if it must wait for program to finish primary output before releasing secondaries
3	41:1	OQUIGN	Value 1, if and only if this output should be ignored
3	40:1	OQUFMT	Value 1, if and only if this message needs a format
3	39:17	OQUDAY	MCS date stamp of message -MM DD YY format
3	22:23	OQUTIM	MCS time stamp when message was generated (in sixtieths of a second)
4	47:24	OQULST	Valid only on a base record and only if OQUFRT is not equal to 1. Equal to OQUNXT of next-to-last record of message (where the base record is the last record)
4	23:24	OQUIML	Link to the input that generated this output
5	47:24	OQUFST	Valid only on a base record and only if OQUFRT does not equal 1. Link to the first record of the message.
5	23:24	OQUNXT	If a base record, link to next base record; otherwise, link to next record in the message. On next-to-last record (where the base is the last record), equals OQULST of the base record
6	47:8	OQUREC	Record number of a record within a message
6	39:16	OQUSMN	Station's output message number by day
6	23:24	OQULNK	Link to all outputs from an input
7	47:24	OQUFPT	Valid only if OQUFMT = 1. Pointer to the format for this message
7	23:24	OQUBWL	Backward link used by recovery
8	47:39	OQUDBSN	Data Base Sequence Number
8	8:8	OQUPRO	PROINDEX of the program that originating this message
8	0:1	OQURCL	Value 1, if and only if this message is the result of a recall request
9	47:24	OQUCNX	Index to conversation of station
9	23:8	OQUTRM	Terminal type of station originating the associated input message.
9	15:1	OQUDEL	Value 1, if and only if a trailer was deleted from this MTS output
9	14:1	OQUFMC	Value 1, if and only if this is a FORMSCOMPOSE message
9	13:11	-----	Not defined
9	2:3	OQUSYS	System index for station

## SYSTEM FORMAT FILES

The Data Communications System Format file contains nearly all the format-related information which the system maintains. This includes a table of format names/locations, a record for each function, and a record for each format. The vast majority of this data is generated when the user's Transaction Control Language (TCL) description of the system is interpreted by the TCL processor. Data in this file may be changed, deleted, or added to during the running of the Message Control System (MCS) by the use of certain system definition messages.

The file name is "DATACOM/QUEUE/FORMATS." It has 30-word records. The MCS uses logical I/O to read the Format file. Following is the layout of the Format file.

Word	Bits	Contents
RECORD 0 – CONTROL RECORD.		
0	47:16	Number of functions
0	31:16	Record number of first function
0	15:16	Record number of first format <control request list> ; <control request>
1	47:16	Number of formats in file
1	31:16	Record index of first name dictionary record
1	15:16	Number of names in name dictionary
2	47:16	Next available patch record
2	31:16	First patch record
3	15:16	Maximum new formats
FUNCTION RECORD LAYOUT.		
0	47:3	4"1" to indicate that this is a function record
0	44:1	Not used
0	43:2	External type: 0 – unedited 1 – integer 2 – ALPHA
0	41:2	Internal type: 0 – unedited 1 – integer 2 – ALPHA
0	39:8	Number of logical records in function (binary value)
0	31:8	Number of translate pairs in function (binary value)
0	23:8	Word index of first pair (binary value)
0	15:16	Logical record number of next function. Zero indicates no next function.
1	47:16	Logical record number of previous function. Zero indicates no previous function.

Word	Bits	Contents
1	31:16	Record index of patch to this function.
1	15:8	Length (in bytes) of function name
1	7:N	(Where N is length from [15:8] above); contains function name
P		(Where P is index from [23:8] of word 0 of this record above); contains <external string> of first translate pair.
P + 1		Contains <internal string> of first translate pair
P + 2		Contains <external string> of second translate pair
P + 3		Contains <internal string> of second translate pair
etc...		Records associated with a particular function are contiguous. The first two words of each such record are identical. The second and subsequent records for a function may not contain the entire name; translate pairs begin in word 2.

#### FORMAT RECORD LAYOUT.

0	47:3	4"2" to indicate that this is a format record
0	44:1	Not used
0	43:1	Residence bit; TRUE means this format should be kept in an array.
0	42:1	Paged bit; TRUE means this format is a multipage format.
0	41:1	Update bit; TRUE means this format is used for update-paging and requires an accompanying unformat.
0	40:1	Unformat bit; TRUE means this is an unformat.
0	39:8	Number of lines in page.
0	31:16	Byte index of first string. ZERO=NULL.
0	15:16	Bytes of data required by this page of a sequential format
1	47:16	Highest-byte index required by format
1	31:8	Number of logical records in format (Binary value)
1	23:16	Space required if used on input
1	7:8	Mask of variables on which to repeat page



Word	Bits	Contents
2	47:16	Pointer to next page
2	31:16	Pointer to previous page
2	15:8	Page number of this format
2	7:8	Number of pages
3	47:16	Link to unformat used to format message upon return from screen This field present only if update bit is on.
3	31:1	This is the sequential bit. If on, it means that this is a sequential format.
3	30:1	If on, means there is a corresponding input format
3	29:16	Not used
3	23:16	Largest upper bound of ([1].[47:16]) of all pages
3	7:8	Contains length of format name
4	47:16	Record index of patch to this format
4	31:16	Original record index for this format
4	15:16	Record index of tested patch to this format
5	47:16	Space required on output
5	31:16	Byte index of the end of the local declaration instructions (variable initialization)
5	15:16	Input record format key
6	47:16	Byte index of first editing instruction. ZERO = NULL.
6	31:16	Byte index of first string of page break code. ZERO = NULL
6	15:1	Set only if page break has occurred
6	14:4	Not used 6 10:5 Repeat index 6 5:1 Set only if format uses headers 6 4:5 Update line count 7 47:N Name of format, where N is the number of bytes indicated by field [3].[7:8]

Next Byte  
(The first byte of  
this segment is  
indicated by field  
[6].[47:16])

This marks the beginning of the editing instructions segment. This segment contains approximately one editing instruction for each editing phrase used in the format declaration. Editing instructions consist of a 1-byte operation (OP) code followed by an operand field that varies in length depending on the operation code. The operand field is always an integral number of bytes in length.

If necessary, this segment can overflow into the next and subsequent logical records without regard to record boundaries. The first byte of a continuation record is considered contiguous to the last byte of the preceding record.

An OP code of 4"FF" marks the end of this segment.

A list of OP codes showing their meaning and associated operands appears after the following discussion of the string segment.

Next Byte  
(after the 4"FF"  
marking the end of  
the editing  
instructions segment.  
The first byte of  
this segment is  
indicated by field  
[0].[31:16])

This is the beginning of the third and last segment of the format record. This is the string segment, and it is very simple in its structure. It consists of all the strings which appeared in the format declaration. The sequence remains unchanged, and the strings are simply catenated to one another with no delimiters.

Like the editing instructions segment before it, the string segment may overflow from one logical record to the next without regard to record boundaries.

<b>OP Code</b>	<b>Editing Phrase</b>	<b>Operands</b>	<b>Comments</b>
0	AW,W<256	W,1 BYTE	Represents ALPHA editing phrase with fixed field width of less than 256
1	IW,W<256	W,1 BYTE	Represents integer editing phrase with fixed field width of less than 256
2	XW,W<256	W,1 BYTE	Represents X (skip) editing phrase with fixed field width of less than 256
3	@L,L<256	L,1 BYTE	Represents a location specifier which specifies a location of less than 256
4	<string> <256	1BYTE LENGTH	Represents a string of less than 256 characters in length. The string itself is stored in the string segment
5	BW,W<256	W,1 BYTE	Represents an ALPHA/TAB editing phrase with a fixed field width of less than 256
6	JW,W<256	W,1 BYTE	Represents an INTEGER/TAB editing phrase with a fixed field width of less than 256
7	A (D, W) W<256	D,1 BYTE W,1 BYTE	Represents an ALPHA editing phrase with a variable field width. There are two operand fields of one byte each. The first operand (D) is the delimiter of the field. The second operand (W) is the internal field width, which is less than 256.
8	I(D,W)	D,1 BYTE W,1 BYTE	Represents an integer editing phrase with a variable field width. Fields are the same as in preceding ALPHA editing phrase.

<b>OP Code</b>	<b>Editing Phrase</b>	<b>Operands</b>	<b>Comments</b>
9	X(D)	D,1 BYTE	Represents an X (skip) editing phrase with a variable field width. D is the delimiter.
10	B(D,W) W < 256	D,1 BYTE W,1 BYTE	Represents an ALPHA/TAB editing phrase with a delimiter used instead of the normal tab character. The internal size is W, which is less than 256.
11	J(D,W) W < 256	D,1 BYTE W,1 BYTE	Represents an INTEGER/TAB editing phrase with a delimiter used instead of the normal tab character. The internal size is W, which is less than 256.
12	T(F,E.W)	F,1 BYTE E,VARIABLES W,1 BYTE	Represents a call on a translate function. The first of the three operands (F) is an index into an array row representing the function. The second operand itself represents an editing phrase at least two bytes long. The length is determined by the operation code in the first byte. The third operand (W) is the internal width, which is less than 7.
13	[	LN,2 BYTES	Byte number of matching "]" ; delimits scope
		PD,2 BYTES	Column to which destination pointer should point upon exit
		MC,2 BYTES	Maximum number of characters between "[" and "]"
		ND,1 BYTE	Number of different delimiters between "[" and "]"

OP Code	Editing Phrase	Operands	Comments
		NM,1 BYTE	Total number of times delimiters are used
		NF,1 BYTE	Number of editing phrases between "[" "]"
		DL, VARIES	Delimiter list (DELIMS), ND 2-byte rows
		DM, VARIES	Delimiter mappings (DELIMAP), NM 2-byte rows
		FD, VARIES	Field mappings (FIELDNO), NF 4-byte rows.
			Immediately following the above files are NF editing phrases followed by "]".  (Refer to the Example at the end of this table.)
14	]	NONE	Used to signal an exit from the "VARIABLE SEQUENCE MODE"
15	AW,W>255	W,2 BYTES	Same as OP code 0 but with 2-byte W for W > 255
16	IW,W>255	W,2 BYTES	Same as OP code 1 but with 2-byte W for W > 255
17	XW,W>255	W,2 BYTES	Same as OP code 2 but with 2-byte W for W > 255
18	@L,L>255	L,2 BYTES	Same as OP code 3 but with 2-byte W for W > 255
19	<string>	T 2 BYTE LENGTH	Same as OP code 4 but with 2-byte W for W > 255
20	BW,W>255	W,2 BYTES	Same as OP code 5 with 2-byte W for W > 255
21	JW,W>255	W,2 BYTES	Same as OP code 6 but with 2-byte W for W > 255

<b>OP Code</b>	<b>Editing Phrase</b>	<b>Operands</b>	<b>Comments</b>
22	A(D,W) W > 255	D,1 BYTE W,2 BYTES	Same as Op code 7 but with 2-byte W for W > 255
23	I(D,W) W > 255	D,1 BYTE W,2 BYTES	Same as OP code 8 but with 2-byte W for W > 255
24	---	---	Not used
25	B(D,W)	D,1 BYTE W,2 BYTES	Same as OP code 10 but with 2-byte W for W > 255
26	J(D,W)	D,1 BYTE W,2 BYTE	Same as OP code 11 but with 2-byte W for W < 256
27	CW,W < 256	W,1 BYTE	Represents C editing phrase with fixed field width of less than 256
28	C(D,W) W < 256	D,1 BYTE	Same as OP code 27 but with delimiter D.
29	CW,W > 255	W,2 BYTES	Same as OP code 27 but with 2-byte W for W > 255
30	C(D,W) W < 255	D,1 BYTE W,2 BYTES	Same as OP code 28 but with 2-byte W for W > 255
31	RPT(N,V) N < 256	N,1 BYTE V,1 BYTE	Open repeat part, max N repetitions Variable index. 255 = no index; 254 = REM.
32	CLSRPT(V)	V,1 BYTE S,2 BYTES I,2 BYTES	Close repeat part. V is update variable index or 255. S and I are string and instruction pointer adjustments if repeat required.
33	@ + L,L < 255	L,1 BYTE	Represents a location specifier increment with increment less than 256
34	@ + L,L > 255	L,2 BYTES	Represents a location specifier increment with increment greater than 255
35	@ - L,L < 255	L,1 BYTE	Represents a location specifier decrement with decrement less than 256

OP Code	Editing Phrase	Operands	Comments
36	@-L,L>255	L,2 BYTES	Represents a location specifier decrement with decrement greater than 255
37	VARASSN (V,L)	V,1 BYTE L,1 BYTE	Assign initial value to variable. Variable index Length of initial value in source text
38	F(W,D) W<256	W,1 BYTE D,1 BYTE	Represents FIXED FIELD editing phrase with fixed field width of less than 256 and decimal field width less than 256.
39	F(W,D) W>255 D>255	W,2 BYTES D,2 BYTES	Represents FIXED FIELD editing phrase with fixed field width of greater than 255 and decimal field width greater than 255.
40	SGN		Indicates leading SIGN for the following fixed field.
41	DLR		Indicates leading DOLLAR SIGN for the following fixed field.
42	CMA		Indicates COMMAS to be inserted into (output), or deleted from (input), the following fixed field.
43	SCM (C, IPI, SPI, EI, CCM)	C,1 BYTE IPI,2 BYTES SPI,2 BYTES EI,N BYTES CCM,5 BYTES	Indicates START CASE. For each case C and the ELSE case, there are offsets to the next possible case (IPI, SPI), N number of editing phrases (EI) for the case, and an end-of-case (CCM) indicator.

<b>OP Code</b>	<b>Editing Phrase</b>	<b>Operands</b>	<b>Comments</b>
44	CCM (IPI, SPI)	IPI,2 BYTES SPI,2 BYTES	Indicates CANCEL CASE mode for current case value. IPI and SPI direct how much to increment editing and string pointers to the end of the case
46	SCMVARV (C, IPI, SPI, IE, CCM)	V,1 BYTE	Indicates start of VARIABLE CASE MODE. Same as 43 except case is repeated on value of variable V.
47	BRKV	V,1 BYTE	Indicates PAGE BREAK on value of variable V.
48	EOL		Indicates INCREMENT LINE COUNT. Page break taken if linecount > page size.
49	HDRP(L)	L,1 BYTE EI,N BYTES	Indicates new HEADER consisting of text produced by EI's at this level, L. Previous headers at this level and lower levels (level 3 is lower than level 2) are no longer in effect.
50	HDRE		Indicates END of editing phrases used to build header.
51	HDRM(L)	L, BYTE	All headers at this level and below are no longer in effect.
52	SGNE		Indicates trailing SIGN for the following field.



Due to the complexity of the way in which variably sequenced fields are represented, the following example is included for clarification.

\* \* \*

Suppose that imbedded within a format declaration are the following:

[A(" ",4), I("#",5), X2, A3, A(" ",4), A(":",3), A(" ",3), A2]

Suppose that the OP code "13" representing "[" is encountered in byte 25 of the format record array; the table below shows exactly how everything between [ & ] will be represented (assuming that the destination pointer is initially on column 1).

BYTE										
ROW	0	1	2	3	4	5	6	7	8	9
20						13	0	106	0	25
30	0	31	4	5	8	"="	1	"#"	2	"*"
40	3	" :	4	0	1	0	2	5	5	0
50	6	0	7	0	1	0	8580	0	5	0
60	8881	0	0	0	9181	0	10	0	9380	0
70	13	0	9580	0	17	0	9880	0	20	0
80	10111	0	23	0	10480	7	2=2	4	8	"#"
90	5	2	2	0	3	7	"*"	4	7	" : "
10	3	7	"*"	3	0	2	14	4	7	" : "

Bytes 35 through 84 are loaded into three arrays as shown below.

DELIMS		DELIMAP			FIELDNO			EDITING PHRASES		
DEL	LNK1	LNK2	FLD		DCOL	ICOL	FFN	CODED		
15:8	7:8	LINKS	15:8	7:8	LINKS	31:16	15:15	0:1	BYTE	CODE SYMBOLIC
=	1	1	0	1	1	1	85	0	85:	7=4 A("=",4)
#	2	2	0	2	2	5	88	1	88:	8#5 I("#",5)
*	3	3	5	5	3	13	95	0	95:	7*4 A(" ",4)
:	4	4	0	6	4	17	98	0	98:	7:3 A(":",3)
		5	0	7	5	20	101	1	101:	7*3 A(" ",3)
					6	6	91	1	91:	22 X2
					7	10	93	0	93:	03 A3
					8	223	104	0	104:	C2 A2

Suppose that the source pointer is set at the beginning of the following message when the "[\" is encountered in the above format.

Source message: 111\*222#3344455 = 66\*77

After formatting, the destination message will look as follows:

Destination message: 55 00222444111 66 77

\* \* \*

## SYSTEM LOG FILES

In order to help analyze all messages received and transmitted in a particular set of disk file queues, the Utility program may be instructed to extract particular information from the queues and put it in a more usable form. The information is taken from an old set of disk file queues and moved into queue log files. The particular type of log created is determined by the contents of the <control statement> during the Utility run. A separate file is built for input and output messages. Since an individual input or output log can be requested from the Utility program, only the log requested produces a file. The user may write a program to read and write the log files in any format desired. All information in the log files is in EBCDIC.

## INPUT LOG

The log records of an Input disk file consist of one record type zero, which, for each Input queue, is followed by: one record type one followed by one record type two, for each message in that Input queue, followed by one record type three indicating the end of this queue. The final record is a record type nine.

If an error is detected during generation of type two records, a record type four is generated indicating the error, and the log of the Input queue in error is discontinued with a record type three.

The record size is 108 characters, 10 records per block, EBCDIC. The title is DATACOM/QUEUE/INPUTLOG. Following is the format of the Input Log records.

**RECORD TYPE ZERO.** This is the first record in the file. It contains the total input disk file information.

Character	Description
1	Record type, value 0
2 - 6	Start queue time in hours and minutes
7 - 12	Start queue date in MMDDYY format
13 - 17	Stop queue time in hours and minutes
18 - 23	Stop queue date in MMDDYY format
24 - 26	Number of Input queues, base 1
27 - 30	Number of stations, base 1
30 - 108	Not defined

RECORD TYPE ONE. There is one record of Control information for every Input queue in the Input disk file.

Character	Description
1	Record type, value 1
2	Record suffix, value 0
3 - 5	Input queue number
6 - 7	System number
8 - 15	Starting record key of input queue, base 0 (BOTBOT)
16 - 33	Not defined
34 - 35	Number of characters in Input queue name
36 - 52	Input queue name
53 - 108	Not defined

RECORD TYPE TWO. There is one record for every message in the Input queue.

Character	Description
1	Record type, value 2
2	Record suffix, value 0
3 - 5	Destination program number. (Refer to DESTINATION PROGRAM NUMBER FIELD below.)
6 - 13	System message number
14 - 19	Originating station number
20 - 26	Time, in sixtieths of a second, that the complete input message was received
27 - 33	Length of message in characters
34 - 41	Base record key of message in Input

Character	Description
42 - 43	Number of valid characters in following field. Maximum field length of message text or 65 characters
44 - 108	Original input message, excluding routing header (if any).

RECORD TYPE THREE. There is one trailing Control record for every Input queue.

Character	Description
1	Record type, value 3
2 - 5	Number of messages unprocessed in Input queue
6 - 10	Total number of messages in Input queue
11 - 108	Not defined

RECORD TYPE FOUR. There is one record for each error. Only one error for each Input queue is allowed.

Character	Description
1	Record type, value 4
2	Error type:
	Value 0 – bad base record
	Value 1 – parity on read
	Value 2 – EOF on read
3 – 8	Base key of error record, base 0
9 – 108	Not defined

RECORD TYPE NINE. This is the last record in the file; it contains the total Input disk file information.

Character	Description
1	Record type, value 9
2 – 7	Total messages
8 – 108	Not defined

DESTINATION PROGRAM NUMBER FIELD. This field normally contains the program number of the program that processed the input message. Some input messages are handled by the system, not by a particular program, and they are assigned a unique program number.

Following are the system-handled input message program numbers.

1. In Error Input queue.

Number	Description
900	Terminate Error NDL result message
901	Input from unknown station
902	DCP fault result message
903	Unknown station event message

2. In Control Input queue.

Number	Description
800	System control message
864	Dollar sign input message
865	Return message to originator
866	Object program message
868	Compute message
869	Mode message
870	Assign message
871	Close message

Number	Description
873	Change message
876	Quit message
877	Intercept message
878	Refresh message
879	Switch message
880	Call out "canned message" input message
881	Log-on message
882	Log-off message
883	Batch program message
890	Badly formatted message
900	Intercepted input message to output
Dial-in	Handle dial-in identification
Sta #	
+ 100	

### 3. In Return Messages Input Queue.

Number	Description
865	Return message from another system
890	Send to system monitors

### 4. In Transmission Error Input Queue.

Number	Description
910	Transmission error result message

### 5. In Invalid Messages Input Queue.

Number	Description
920	Invalid message key
921	Security violation
923	Invalid modify attempt
924	Station is busy

## OUTPUT LOG

The log records of an Output disk file consist of one record type zero, which, for each Station queue, is followed by: one record type one, followed by one record type two for each message in that station queue, followed by one record type three indicating the end of this queue. The final record is a record type nine.

If an error is detected during generation of type two records, a record type four is made, indicating the error, and the log of the station queue in error is discontinued with a record type three.

The record size is 108 characters, 10 records per block, EBCDIC. The title is DATACOM/QUEUE/OUTPUTLOG. Following is the format of the Output Log records.

RECORD TYPE ZERO. This is the first record in the file. It contains the total output disk file information.

Character	Description
1	Record type, value 0
2 - 6	Start queue time in hours and minutes
7 - 12	Start queue date in MMDDYY format
13 - 17	Stop queue time in hours and minutes
18 - 23	Stop queue date in MMDDYY format
24 - 26	Number of Input queues, base 1
27 - 30	Number of stations, base 1
31 - 108	Not defined

RECORD TYPE ONE. There is one record of Control information for every Station queue in the Output disk file.

Character	Description
Suffix 0:	
1	Record type, value 1
2	Record suffix, value 0
3 - 9	Station number
10 - 11	System number
12 - 19	Starting record key of station queue, base 0
20 - 22	Station poll characters, left-justified
23 - 25	Station select characters, leftjustified
26 - 108	Not defined
Suffix 1 - 5:	
1	Record type, value 1
2	Record suffix, value 1
3 - 5	Length of following field including continuation record type ones, if any
6 - 108	Station name
1	Record type, value 1
2	Record suffix, value 2-5
3 - 108	Continuation of station name
Suffix 6:	
1	Record type, value 1
2	Record suffix, value 6
3 - 5	Length of following field including continuation record type ones, if any
6 - 108	Station remarks

RECORD TYPE TWO. There is one record for every message in the station queue.

Character	Description
Suffix 0:	
1	Record type, value 2
2	Record suffix, value 0
3 - 10	System message number
11 - 15	Station output message number
16 - 22	Time complete output message made in sixtieths of seconds
23 - 34	DBSN
35 - 37	Program Number
38 - 44	Length of message in characters
45 - 52	First record key of message, base 0
53 - 60	Base record key of message, base 0
61 - 62	Number of valid characters in following field. Maximum field length of message text or 46 characters.
63 - 108	Message text, excluding routing header and title (if any).

RECORD TYPE THREE. There is one trailing Control record for every station queue.

Character	Description
1	Record type, value 3
2 - 4	Number of unprocessed messages in Station queue
5 - 9	Total number of messages in station queue
10 - 108	Not defined

RECORD TYPE FOUR. There is one record if a bad record is encountered in the Station queue.

Character	Description
1	Record type, value 4
2	Error type: Value 0 - bad base record Value 1 - parity on read Value 2 - EOF on read
3 - 8	Base key of error record, base 0
9 - 108	Not defined

RECORD TYPE NINE. This is the last record in the file. It contains the total Output disk file information.

Character	Description
1	Record type, value 9
7	Total messages
8 - 108	Not defined

## AUDIT LEDGER FILE

The Audit Ledger file is a history of the archival dump tapes which were created for the data communications environment. The record size is 10 words, and the block size is 30 words. Following is the layout of the Audit Ledger file.

Word	Description
0	Serial number of the archival dump tape
1	System number as declared in the TCL
2	System index
3	Sequence control number
4	Start date (generating date)
5	Start time (generation time)
6	Stop date (dump date)
7	Stop time (dump time)
8-9	Unused

## ARCHIVAL AUDIT TAPE FILE

The Archival Audit tape file contains the archival audit dump used during archival recall and archival recovery. Following is the layout of the Archival Audit file.

RECORD ZERO. This contains dates and times of the Archival period.

Word	Description
1	Start date (generation date)
2	Start time (generation time)
3	Stop date (dump date)
4	Stop time (dump time)
5-60	Unused

RECORD ONE. This is a sentinel record marking the beginning of the tape used for Archival recall.

Word	Description
1	"SENTIN"
2	"AL REC"
3	"ORD RE"
4	"CALL "
5-60	Unused

RECORDS TWO – N. The Output queues are dumped here by station. All output messages released to a terminal are dumped in Output disk file format. A description of these records appears under OUTPUT QUEUE FILE in this appendix.



RECORD N+1. This record contains file size information. It is a sentinel record marking the end of the records used for archival recall and the beginning of the system files used during archival recovery.

Word	Bits	Description
1	---	"SENTIN"
2	---	"AL REC"
3	---	"ORD SY"
4	---	"STEM "
5	47:16	Number of records in the Control file
	31:16	AREASIZE of Control file
	15:16	Record size of the Control file
6	47:16	Number of records in the Input file
	31:16	AREASIZE of Input file
	15:16	Record size of the Input file
7	47:16	Number of records in the Output file
	31:16	AREASIZE of Output file
	15:16	Record size of the Output file
8	47:16	Next available patch record in Format file
	31:16	Unused
	15:16	Record size of the Format file
9	47:48	Value 1 – indicates duplicate files
10-60	---	Unused

RECORDS N+2 to N+X. (Where X equals the number of Control file records, plus the number of Format file records, plus one record for each Input queue, plus one record for each Output queue, plus 4.) The file contains (in this order the entire Control file, Record zero of the Input Queue file, a skeletal Input Queue file (one record per Input queue), a skeletal Output Queue file (one record per Output queue), and the entire Format file. The format of each record is described in appendix B under the appropriate file type.

RECORD N+X+1. This sentinel record marks the end of the records used to store the system files used during archival recovery and the beginning of the records which indicate that nonrecoverable queue messages are to be reprocessed.

Word	Description
1	"SENTIN"
2	"EL REC"
3	"ORD AR"
4	"CHIVE "
5	"NONRCV"
6-60	Unused

RECORD N+X+2 . This record contains the archive records for nonrecoverable queues to be reprocessed during an archival recovery.

RECORD  $N + X + 3 + J$  . (Where J equals the number of nonrecoverable queue archive records). This sentinel record marks the beginning of the records which indicate that recoverable queue messages are to be reprocessed.

Word	Description
1	"SENTIN"
2	"EL REC"
3	"ORD ARE"
4	"CHIVE"
5	"RCV"
6-60	Unused

RECORD  $N + X + 4 + J$  to END. These are archive records for recoverable queues to be reprocessed during an archival recovery.



## APPENDIX C

# THE CONTROL WORDS ARRAY

CONTROLWORD[0] indicates the results of a GETMESSAGE or SENDMESSAGE call.

Following is a list of the GETMESSAGE values.

Value	Meaning
0	Good result, more to message.
1	Good result, end of message.
64	Good result, overlap I/O started, awaiting I/O complete.
128	Bad result, source error.
129	Bad result, read parity.
130	Bad result, read EOF.
131	Bad result, message record error.
132	Bad result, file not ready.
133	Bad result, unexplained I/O error.
134	Bad result, invalid request.
135	Bad result, insufficient data for request.

Following is a list of the SENDMESSAGE values.

Value	Meaning
0	Good result.
64	Good result, destination marked down, no alternate STA.
128	Bad result, destination error.
129	Send to Area denied, because there are currently no stations in the area.
130	Bad result, data error.
131	Bad result, originator error.
132	Bad result, nonqueued output to NOP station.
133	Bad result, invalid log-on or log-off attempt by a non Access Control Program.
136	Undefined pass of control.
137	Received in response to a TP-to-TP and WAIT request. Destination TP inoperative or aborted while processing the transaction.
138	Received in response to a TP-to-TP and WAIT request. The TP-to-TP response was larger than the Common area of the originating TP.
143-148	Access Control Violation – returned only to an Access Control Program.
143	Undefined access key.

Value	Meaning
-------	---------

- |     |                                       |
|-----|---------------------------------------|
| 144 | Access key not allowed at station.    |
| 145 | Too many users.                       |
| 146 | User already logged on.               |
| 147 | Someone else already logged on.       |
| 148 | Station is assigned (log-off denied). |

CONTROLWORD[1] is used to control the actions of GETMESSAGE.

CONTROLWORD[1].[47:12] is not defined.

CONTROLWORD[1].[35:3] is used to specify the type of recall message (valid only if CONTROLWORD[1].[7:8] = 2). Value 0 is standard; value 1 is a recall by date (archival recall); and value 2 recalls the last n messages.

CONTROLWORD[1].[32:1] is used by the MCS to inform the Recall program whether the title should be inserted in the recalled message (valid only if CONTROLWORD[1].[7:8] = 2). Value 0 tells the program not to insert the title; value 1 tells it to input the title.

CONTROLWORD[1].[31:8] controls read overlap. Its values are as follows:

Value	Meaning
0	Wait for I/O complete while getting a message.
1	Return with result of 64 if I/O was started but not completed. Valid only for Input queue source. The Process program can make the identical GETMESSAGE call at a later time to see if the I/O is complete, or it may wait on its input event which is caused when the I/O is complete.

CONTROLWORD[1].[23:8] controls message recall requests. The values are as follows:

Value	Meaning
0	Input messages by system message number; valid for station or Input queue source.
1	(Invalid request control).
2	(Invalid request control).
3	Output messages by system MSG number; valid only for station source.
4	Output messages by station output message number; valid only for station source.
5	Output messages by time of day; valid only for station source.
6	Input/Output messages by system message number; valid only for station source.
7	Input/Output messages by station message number; valid only for station source.
8	Input/Output messages by time of day; valid only for station source.

CONTROLWORD[1].[15:8] contains the message source type. Value 0 means that CONTROLWORD[7] is an Input queue number. Value 1 indicates that CONTROLWORD[7] is a station number.

CONTROLWORD[1].[7:8] controls the GETMESSAGE functions. The values are as follows:

Value	Meaning
0	Read bottom message in my Input queue and return it in the queue passed from GETMESSAGE to the MCS.
2	Recall processed input and/or output messages. <ol style="list-style-type: none"> <li>CONTROLWORD[1].[15:8] is source type.</li> <li>CONTROLWORD[7] is the source number</li> <li>CONTROLWORD[1].[23:8] is Recall Control</li> <li>CONTROLWORD[1].[35:3] is recall type.</li> <li>CONTROLWORD[1].[32:1] is 1 if title required.</li> </ol> <p>The message queue contains information about the messages to be recalled. The contents of the queue are documented at the end of this appendix.</p>
3	Return station bits. CONTROLWORD[1].[15:8] must be set to 1, and CONTROLWORD[7] is the station number. The bits are returned in CONTROLWORD[9].
4-7	Not defined.
8	Return name of source. CONTROLWORD[1].[15:8] is source type. CONTROLWORD[7] is the source number. The message queue contains the name in input record format.

CONTROLWORD[2] controls the SENDMESSAGE functions.

CONTROLWORD[2].[47:16] is not defined.

CONTROLWORD[2].[31:8] controls the line feed and carriage return characters sent at the end of a message to an unbuffered terminal. Its values are as follows:

Value	Meaning
0	Add CR,LF,LF to end of message.
1	Add CR to end of message.
2	Add LF,LF to end of message.
3	Suppress CR,LF,LF at end of message.

CONTROLWORD[2].[23:2] is not used.

CONTROLWORD[2].[21:1] is used to control the type of log-off message. Value 0 designates a normal log-off message; value 1 is a system log-off message.

CONTROLWORD[2].[20:1] is set to value 1 if and only if format pointers are already set up and exist as CONTROLWORD[17].[23:24]. This word is set and used by SENDMESSAGE when processing RECALL messages.

CONTROLWORD[2].[19:1] controls the routing header format of output messages. The original message format is <routing header>.<text>. The message format is <routing header> CR LF <text>. Value 0 means: Use the destination to determine if the format to be used is a return or original format. If the originator of the message is not equal to the destination, use the original message format; otherwise, use the return message. Value 1 designates that the message format is to be in a return message format, and it is valid only when the first piece of message is sent.

CONTROLWORD[2].[18:1] controls the routing header to be used with the outgoing message. Following are the values for this word.

Value	Meaning
0	Use saved routing header with message.
1	User is supplying routing header; valid only when the first piece of message is sent. Routing header is delimited by a period (.) and is included in the message length as specified in CONTROLWORD[3].

CONTROLWORD[2].[17:2] controls the title to be attached to the output message. The values are as follows:

Value	Meaning
0	Use destination to determine title format.
1	Do not add title to MSG.
2	Multiple station output title.

CONTROLWORD[2].[15:3] controls the SENDMESSAGE functions. The values are as follows:

Value	Meaning
0	Accumulate and send a message to a destination.
1	Enable request.
2	Disable request.
3	Validate destination.
4	Acknowledge originating station. CONTROLWORD[2].[12:5] = 0 (station by number). CONTROLWORD[10] contains the station number.
5	Log-on message.
6	Log-off message (refer to CONTROLWORD[2].[21:1]).
7	Cancel accumulate and send request, if one is started.

CONTROLWORD[2].[12:5] is the destination control. It dictates what is affected by the request. If an area, station, Input queue, or program number is required, it must be inserted in CONTROLWORD[13]. If an area, station, or Input queue name is required for validation, it should appear in the text area. If a station name is required for a send request, it should appear in CONTROLWORD[22]. Following are the values for CONTROLWORD[2].[12:5].

Value	Meaning
0	Request is for station by number.
1	Request is for area by number; valid only for accumulate and send requests.
2	Request is for program by number or by message key; valid only for accumulate and send requests. Primary response goes back to station.
3	Request is for system monitors; valid only for accumulate and send requests.
4	Request is for System Network Control stations; valid only for accumulate and send requests.
5	Request is for Input queue by number; valid only for enable or disable requests.
6	Request is for stations that will receive service message; valid only for accumulate and send requests.
7	Request is for program by number or by message key; valid only for accumulate or send requests. Primary response comes back to program.
8	Request is for station by name; valid only for enable, disable, and validation requests.
9	Request is for area name; valid only for validation requests.
10	Request is for station by name; valid only for accumulate requests.
11-12	Not defined.
13	Request is for Input queue by name; valid only for enable and disable requests.

CONTROLWORD[2].[7:8] is used when the request is to accumulate a message and send it to a destination. The values are:

Value	Meaning
0	First or intermediate part of a message is being supplied.
1	Final part of message supplied. Start message to destination but retain base record so that the message can be sent to another destination.
2	Final part of secondary message supplied.
3	Final part of primary message supplied.

CONTROLWORD[3] indicates the length of the message part to be accumulated by SENDMESSAGE into a complete message. If the length is less than 0, the message is sent directly to the destination station without being audited on disk or accumulated into a complete message, and the absolute value of the length would be 1 greater than the true length of the message part. Because the message is not being audited, if the station is not ready to receive the message on the first try, the message is lost.



CONTROLWORD[4] contains data about the message obtained by GETMESSAGE. It is valid only after a request to read a message.

CONTROLWORD[4].[47:4] indicates how the message got here. Value 0 means it is a normal message with a message key. Value 1 indicates that the message has no message key and that it is from an assigned station.

CONTROLWORD[4].[43:1] designates the origin of the message. Value 0 indicates that the originator is a data communication station, and value 1 means it is an Object job.

CONTROLWORD[4].[42:3] designates the type of input message. The values are:

Value	Meaning
0	Normal message from originator.
1	Message generated from application program; the response is going back to the station at which the original input was entered.
2	Message generated by a control message.
3	Message generated by an application program; the response is going back to that program.
4-6	Not defined.
7	Flush recovery indicator.

CONTROLWORD[4].[38:1] indicates whether the originator must be acknowledged. Value 0 designates that it need not be acknowledged; value 1 indicates that it should.

CONTROLWORD[4].[37:3] is the Transaction Retry-counter. It indicates the number of times this transaction has been submitted to the Process program and caused it to terminate abnormally. It is incremented by 1 whenever an input transaction causes a Process program to abort.

CONTROLWORD[4].[34:2] identifies the Relative Input queue containing this message.

CONTROLWORD[4].[32:8] identifies the number of the Input queue in which this message resides (as defined in the TCL).

CONTROLWORD[4].[24:1] is not used.

CONTROLWORD[4].[23:24] contains the unique system message number assigned to the message at input.

CONTROLWORD[5] is valid only after a request to read a message.

CONTROLWORD[5].[47:24] contains the date of the message obtained by GETMESSAGE in Month, Day, and Year format. Value 120175 would be December 1, 1975. The value of the date is in binary form.

CONTROLWORD[5].[23:24] contains the time of the message obtained by GETMESSAGE in sixtieths of a second.

CONTROLWORD[6] is not used.

CONTROLWORD[7] contains information about the source of the message pertaining to a GETMESSAGE request. If the User program sends a Read Input Queue request, this word is set by GETMESSAGE to indicate the source Input queue number. If the User program sends a Recall request, this word is set by the caller to indicate the station or Input queue number of the source.

CONTROLWORD[8] contains information about the source of the message.

CONTROLWORD[8].[47:1] is set to value 1 if the program to handle this message is disabled.

CONTROLWORD[8].[46:1] is set to value 1 if the Input queue from which the message was obtained by a GETMESSAGE call is under a rerun condition. The rerun condition is controlled by a CHANGE INPUTQUEUE system control message.

CONTROLWORD[8].[45:1] is set to 1 if the program to handle this message is a test program.

CONTROLWORD[8].[44:13] is not used.

CONTROLWORD[8].[31:16] contains the item count for this message.

CONTROLWORD[8].[15:8] usually contains the program number of the program receiving this message obtained by a GETMESSAGE call. The exceptions are when:

1. The program is defined to be a test program. This field would then contain the program number of the second message key.
2. The message is a control message, in which case this field would have a value of 0. CONTROLWORD[8].[7:8] has unique values defining a control message.

CONTROLWORD[8].[7:8] contains the internal program index of the program receiving this message obtained by a GETMESSAGE call. In the case where the program number defines a control message (refer to CONTROLWORD[8].[15:8] above), the values are:

Value	Meaning
65	Return message, routed by routing header.
90	Badly formatted return message: no routing header, invalid characters in routing header, routing header too long.
110	Transmission error message; MSG is a message of NDL error bits.
120	Message with invalid message key.
121	Message with invalid message key; station-to-program security error.
123	Message with modify program key; updates disabled.
124	Message with busy message key.

CONTROLWORD[9], for a return station bits request, contains the station bits for the station specified in CONTROLWORD[7].

CONTROLWORD[10] contains the station number of the originator of the message obtained by GETMESSAGE.

CONTROLWORD [11] contains information associated with the originator of the message. If the message originated from an Object job, this word contains the job number; if the message originated from a Batch job, this word contains the File Relative Station Number (FRSN); if the message originated from a station, this word contains the station number.

CONTROLWORD[12], for a GETMESSAGE call to read a message from an Input queue (CONTROLWORD[1].[7:8] = 3), contains the station bits of the originator of the message. This is where an Access Control module indicates the value to change the station bits during a successful log-on or log-off.

CONTROLWORD[13] must be filled by the Process program to satisfy a SENDMESSAGE call. It contains a station, area, Input queue, or program number as specified in CONTROLWORD[2].[12:5]. This field is set to 0 when an input message is passed to a program. A value of 0 implies that all subsequent responses are sent to the originator of the message. This word affects the destination of the message only at the first pass of a response.

CONTROLWORD[14] is to be altered only if the originator was a Batch or Object job, and it is desired to send a response back to the originator. In this case, move CONTROLWORD[11] here.

CONTROLWORD[15] is set by SENDMESSAGE after a validate station name or validate area name request and consists of the station number or area number.

CONTROLWORD[16] is related to recovery.

CONTROLWORD[16].[47:2] indicates the recovery status of the system. Its values are:

Value	Meaning
0	The system is not in recovery mode.
1	The system is in recovery mode caused by a TP abort.
2	The system is performing an archival recovery.
3	The system is in recovery mode caused by a Halt/Load or an abnormal termination of the MCS.

CONTROLWORD[16].[45:1] when value equals 1 indicates that this program is a test program.

CONTROLWORD[16].[44:1] allows the user to override the UPDATE specification of an output format. Value 0 tells the program to update the message; value 1 tells it to display the message.

CONTROLWORD[16].[43:1] is set to value 1 if the output of this transaction is going to be duplicatedly audited. The user may override the duplicate auditing of the output by setting the bit to value 0. If the duplicate auditing is to be reinstated for this transaction, the user must set the bit back to 1. If duplicate audit files exist, the user can cause any output to be duplicatedly audited by setting this bit to 1.

CONTROLWORD[16].[42:1] is the WAITFORAUDIT bit, as discussed in detail in Section 5. Value 1 indicates that the program must wait for output audit of primary output before exiting from transaction state.

CONTROLWORD[16].[41:1] is the TRANSACTIONSTATE bit, as discussed in detail in Section 5. It is set and reset by the Process program. It is set to value 1 if the program is in transaction state.

CONTROLWORD[16].[39:8] contains the index into the MCS program table.

CONTROLWORD[16].[31:8] contains the index into the MCS Input queue table.

CONTROLWORD[16].[23:24] contains the input disk address of the base record of this input message (to be stored in the restart data set).

CONTROLWORD[17] contains the output message-ID. This indicates which format, if any, is to be applied to this message. This field contains spaces placed by the MCS on an initial pass of control of an input message.

CONTROLWORD[18].[47:8] contains the 1-character ID used to differentiate between users of a multi-user log-on station.

CONTROLWORD[18].[39:39] contains the Data Base Sequence Number.

CONTROLWORD[18].[0:1], when set to value 1, is a recalled message. It is set by the Recall program, GEMCOS/RECALL, and stored in the Output Audit Queue file.

CONTROLWORD[19] is not used.

CONTROLWORD[20] contains the module index associated with the message key.

CONTROLWORD[21] contains the function index associated with the message key.

CONTROLWORD[22] through CONTROLWORD[24] contain the name of the originating station terminated by 4"FF". If the Process program wishes to route output to a station by name, the station's name is placed here, terminated by 4 "FF".

CONTROLWORD[25] contains the first word of the individual identifier for the station that originated the input.

CONTROLWORD[26] contains the second word of the individual identifier for the station that originated the input.

CONTROLWORD[27] through CONTROLWORD[33] contain the routing header of the input message.

CONTROLWORD[34] is used by SENDMESSAGE (but not GETMESSAGE). It should not be changed by the Process program.

## RECALL MESSAGE QUEUE

The message queue for a RECALL message should contain one record whose content depends on the recall type (CONTROLWORD[1].[35:3] and the recall control CONTROLWORD[1].[23:8]). The first word always contains all 4 "FF" (all bits on). Following is a description of the RECALL message queue contents based on recall type and recall control.

Type	Control	Queue Contents
0 (standard)	0,3,4,6,7	There is one word for each message number to be recalled with the message number in field [23:24].
0 (standard)	5,8	There is one word with the starting time in [22:23] and the stopping time in [45:23]. All times are in sixtieths of a second.
1 (archival recall)		There are two words. The first is the time word as described above. The second is the date in [23:24] in the following format: (MONTH*10000) + (DAY*1000) + YEAR where MONTH, DAY, and YEAR are two digits.
2 (recall LAST)		There is one word containing the number of records to be recalled.

GETMESSAGE returns in the message queue the recalled messages in input queue disk file format. Preceding each returned message in the queue is a 2-word tag record whose format is as follows:

WORD[0] = 1 if the following message is an input message  
WORD[0] = 2 if the following message is an output message  
WORD[1] = System or station output message number of message.

The last message in the queue is the original RECALL message updated with a result indicator. The result indicator is an 8-bit field, each bit representing a valid result, and it is contained in field [47:8] of the word in which it resides. If the recall is by message number, each word containing a message number to be recalled contains the result indicator. In all other cases, the result indicator is the first word following the all "FF" word.

The RECALL result indicators are as follows:

<b>Bit Location</b>	<b>Value</b>	<b>Definition</b>
[40:1]	1	Recall number found and returned.
[41:1]	1	Parity in message, message number skipped.
[42:1]	1	Bad record format, message number skipped.
[43:1]	1	End-of-File, message number skipped.
[44:1]	1	Request for I/O recall, Input message bad.
[45:1]	1	Request for I/O recall, Input not available.
[46:1]	1	Too many messages recalled (29 maximum).
[47:1]		Not used.

The recall mechanism within the MCS terminates whenever one of the recall error bits (41-44, 46) must be turned on.



# APPENDIX D

## HOW TO READ THE MCS STATION MONITOR

Monitor code for all activity on a station can be obtained for a station by the following CHANGE command:

```
?CHANGE STATION (15) MONITOR TO TRUE
```

The monitor can be turned on and off at will. Monitor output is written to a printer file which is printed at system EOJ. In order to use this feature, the MONITOR option must be set when compiling the MCS.

When monitoring the activity of a station, the user receives an entry on the monitor file every time certain actions are performed for that station. These action types are:

1. Any DCWRITE affecting the station.
2. Any input from or concerning the station.
3. Any output request for the station.
4. Any output directed to the station. Although it could be part of #1, this is broken out separately.
5. Any output result for a station.

### TYPE 1 ACTION

```
# DCWRITER <dcwrite number> <dcwrite variant> <station TCL number> <station  
internal index> <time> (hrs: min: sec: )
```

Example:

```
# DCWRITER 38 1 36 4 1918035 (8:52:47)
```

The numerals in this example are defined as follows:

1. 38 is the DCWRITE type (change application number).
2. 1 is the variant.
3. 36 is the station number defined in TCL.
4. 4 is MCS index to station tables for this station.
5. 1918035 is the time of the activity.
6. 8:52:47 is the time broken down by component.

### TYPE 2 ACTION

```
# INPUT FOR STA = <station TCL number> <station MCS index>  
LSN - <station lsn in NDL> @ <time> (hrs: min: sec: )  
EBCDIC TRANSLATION OF THE DATA RECEIVED  
HEXADECIMAL LIST OF THE ENTIRE MESSAGE
```



Example:

```
# INPUT FOR STA = 356 4 LSN - 105 @ 1918044 (8:52:47)
????????
050000000069 000000000000 0F0000000000 ID445C000000
000000660000 000000000000 000000000009 000000000000
```

This example is defined as follows:

1. 356 is the station number in TCL.
2. 4 is the MCS index to station tables for this station.
3. 105 is the LSN for the station from NDL.
4. 1918044 is the time of the input.
5. (8:52:47) is the breakdown of the time.
6. ???? is the EBCDIC translation of the message beginning in word 6. There is no valid text in this message.
7. The hexadecimal representation of the message.

Word[0].[47:8] is always the type of input message. In this case, "05" is a good result from a previous output to the station. A type "00" input is a good input message, and the EBCDIC characters would list the actual text of the message. Refer to the DCALGOL manual for other type message results.

## TYPE 3 ACTION

```
# OUTPUT REQUEST, STA = <station TCL number> <station MCS index> LIN = <line number> @ <time> (hrs: min: sec: )
```

Example:

```
# OUTPUT REQUEST, STA = 16 1 , LIN = 2 @ 1925834
(8:54:57)
```

This example is defined as follows:

1. 16 is the station number from TCL.
2. 1 is the MCS index to the station tables for this station.
3. 2 is the MCS line number the station is on.
4. 1925834 is time of activity.
5. (8:54:57) is breakdown of time.

Two type-3 requests could be generated from one output, depending upon whether the message is taken off disk.

## TYPE 4 ACTION

```
# OUTPUT TO STA = <station TCL number> <station MCS index> ,
@ <time> (hrs: min: sec: )
```

Example:

```
# OUTPUT TO STA = 16 1 , @ 1925852 (8:54:57)
```

This example is defined as follows:

1. 16 is the station TCL number.
2. 1 is the station MCS index.
3. 1925852 is the time of output.
4. (8:54:57) is the time breakdown.

## TYPE 5 ACTION

```
# OUTPUT RESULT, STA = <station TCL number> <station MCS  
index>, STO = <station TCL number> <station MCS index>,  
LIN = <line number> SEL = <integer> NOT = <integer>  
@ <time> (hrs: min: sec: )
```

Example:

```
# OUTPUT RESULT, STA = 16 1 , STO = 16 1 , LIN = 2  
SEL = 0 NOT = 1 @ 1925885 (8:54:58)
```

This example is defined as follows:

1. 16 is the station TCL number for station.
2. 1 is MCS index to station tables for this station.
3. 16 same as 1, would be different if alternate station used.
4. 1 same as 2, would be different if alternate station used.
5. 2 line number.
6. 0 tells that everything is ok.
7. 1 number of pieces of message sent.
8. 1925882 time.
9. (8:54:58) breakdown of time.

The normal sequence of events for a station would be:

1. Several DCWRITES reading the station.
2. A type-2 good input for a station.
3. A type-3 output request, possibly two for the station.
4. A type-4 output to the station.
5. A type-5 good result from the write-to station

Monitoring more than one station is possible, but it makes the monitor harder to read because of interspersed data.



# **APPENDIX E**

## **MCS ERROR MESSAGES**

Listed in this appendix are the various messages which are output by the MCS and either "displayed" at a supervisory console or transmitted to a Network Control station. The action to be taken in response to each message is indicated.

### **MESSAGES DISPLAYED ON THE SUPERVISORY CONSOLE**

The following messages are displayed on the supervisory console. The MCS expects an operator response to all messages listed except for:

FINAL ROW: = <integer>  
DCWRITE = 0, RESULT = <integer> TRY AGAIN.

The MCS failed when attempting to initialize its primary queue. Either the NDL does not have the MCS title declared as a legal MCS, or another copy of the MCS is already running. Answering AXYES to the accept message causes the MCS to attempt to initialize primary queue again. Entering any other response causes the MCS to abnormally terminate.

DISK FILES BAD, DO LIST SEE IT.

The three queue files (Input, Output, and Control) that the MCS needs in order to run are not consistent, i.e., were not created together. A response of CONTINUE tells the MCS to ignore this error condition. Any other response causes the MCS to abnormally terminate.

FINAL ROW: = <integer>.

The MCS has just allocated the last available row on its queue files and is going to EOJ so that a REGENERATE can be done. This message does not require an operator response.

INTERCOM FAILED.

The attempt to set up an intercom queue with the controller failed. Any response allows the MCS to continue.

LOOK AT PRINTER.

The MCS has received an erroneous DCWRITE result and is going to EOJ. It has written a printer file with the bad results which caused it to terminate. Any response causes the MCS to abnormally terminate.

## MESSAGES APPEARING AT STATIONS.

The following error messages can appear at any station in response to an input.

### BUSY

This station was declared in the TCL to be one-in/one-out. The message means that a second input was made before output was received from the first input.

### INVALID MESSAGE KEY.

The MCS does not recognize the message key as being a valid one declared in the TCL. The message key must be at the beginning of the message, unless MKEPOSITION was declared for this station in the TCL; and must be delimited by a nonalphanumeric character, unless FIXEDMKE was declared for this station in the TCL.

### SECURITY ERROR.

A user has attempted to input data at a SIGNON = TRUE station without successfully logging on.

### TRANSMISSION ERROR.

Thirty seconds have elapsed since the last character was typed on an unbuffered device, and an ETX has not yet been transmitted. The input for the station is discarded. (This message is NDL-dependent and appears if the suggested GEMCOS request sets are used, as discussed in Section 11.)

### UPDATES DISABLED.

Someone has input a ?DISABLE UPDATES System Control command.

## MESSAGES TO A SYSTEM NETWORK CONTROL STATION (SPO)

The following error messages are printed at a System Network Control station. Certain of the messages are for information only and require no operator intervention, while others require operator action.

DCP <integer> FAULT <integer>

The MCS has been informed of a DCP fault.

**\*\*ERROR INPUT FROM UNKNOWN BATCH JOB <time>**

The MCS has received input from a Batch job which is not following the conventions required for a Batch job. The input is discarded.

**\*ERROR <number> (DLA) <station name> <number> <time>: <text>**

The MCS received an error on the specified station. <Text> contains a description of the problem. The "(DLA)" is only included if there is a line error. In all cases, the offending station is marked down. To recover, the user should first attempt to ENABLE the station. If the error occurs a second time, the BOT pointer should be changed to skip the output causing the error before attempting to ENABLE the station again. Following is a list of some of the common errors (and their respective meanings) which might be responsible for the conditions.

Error Message	Meaning
BAD OUTPUT QUEUE	There is a problem with the MCS output queue for the station.
EXCESSIVE SECURITY ERRORS	Someone has attempted to log on and has failed twice consecutively.
EXCESSIVE TRANSMISSION ERRORS	The MCS has marked the station down rather than waste time handling transmission errors.
OUTPUT TIMEOUT	The MCS sent a message but did not receive a good result within six minutes of sending the output. NDL specifications should be checked. This error only occurs when STATIONTIMEOUT is set for the station.
TRANSMIT ABORT RECEIVE ABORT	This indicates hardware or line problems resulting in type 99 error messages from the DCP.

**\*ERROR, UNKNOWN STATION (more info)**

The MCS received a message in its primary queue from a station not identified as one of the user's stations. The input is discarded.

**\*ERROR, UNKNOWN STATION EVENT (more info)**

The MCS received an unrecognizable DCWRITE type-1 message in its primary queue.

**\*FILE OPEN ERROR (more info)**

An object job attempted to open a remote file which contained stations under the control of the MCS and was rejected. The reason for the rejection is specified.

**\*\* FORMAT ERROR <number> FOR STATION <number>  
IN SYSTEM <number> ON FORMAT <format name>**

The MCS found an error while applying a format declared in the TCL against the data transmitted. The errors are numbered, and their meanings are specified in Section 3.

**\*PROGRAM NOT INITIALIZED (more info)**

The MCS attempted to start the program but failed because of the reason specified in the remainder of the message (parameter mismatch, missing object file, etc.). If this message is received, corrective action should be taken, and the user should DISABLE and CLEAR the program before once again attempting to start the program.

**\*PROGRAM SCHEDULED (more info)**

A program was started by the MCS but has gone into the schedule to be run at a later time.

**\*PROGRAM SUSPENDED (more info)**

A program was started, and the MCS is awaiting MCP or operator intervention to continue running.

**\*PROGRAM TERMINATED (more info)**

A Transaction Processor (TP) has gone to EOJ without receiving a message from the MCS to do so. The reason for the termination and a brief task history are included within the message. Once a new copy of the program is ready to be tested, the user should DISABLE, CLEAR, and ENABLE the program. If the TP terminated while in DMS II transaction state, additional instructions are included in the message to initiate the synchronized recovery cycle (see \* RECOVERY CYCLE WILL BE INVOKED WHEN error message).

**PROGRAM TIMEOUT (more info)**

The MCS has passed an input message to a program, and the number of elapsed seconds exceeds the number of seconds specified in the <timeout statement>. The <timeout statement> is a required parameter of the <program statement> defined in the TCL. This message is a warning only. No direct action is taken, but the MCS continues to inform of this condition every <timeout statement> seconds.

**\*\* <time> DCWRITE# <integer> ERROR FOR STATION <integer>,  
RESULT = <integer>**

The MCS attempted to do the specified DCWRITE for the station listed but was returned a bad result as indicated at the end of the message.

## RECOVERY-RELATED MESSAGES

The following are messages which might be generated during an MCS synchronized recovery cycle.

\*RECOVERY ABORTED FOR SYSTEM <number>  
\*RECOVERY STACK FAILED

The recovery stack failed. After corrective measures have been taken, the recovery cycle should be initiated manually, using a Network Control command.

\*RECOVERY ABORTED FOR SYSTEM <number>  
\*RESTART TP FAILURE

The restart TP either terminated abnormally or passed back an error condition to the MCS. After corrective measures are taken, the recovery cycle should be initiated manually, using the Network Control command.

\*RECOVERY ABORTED FOR SYSTEM <number>  
\*RESTART TP PASSED INVALID RESTART DATA

This message indicates that the queues in use may be invalid. After corrective measures are taken, the recovery cycle should be initiated, using the Network Control command.

\*RECOVERY ABORTED FOR SYSTEM <number>  
\*TP ABORTED DURING RECOVERY CYCLE  
\*RECOVERY WILL BE RE-INITIATED

One of the user TPs received a DMS II abort exception during the recovery cycle and passed this information back to the MCS; however, no program has terminated abnormally. The condition may be caused by a program closing the data base while in transaction state. The MCS terminates the current recovery cycle.

\*RECOVERY ABORTED FOR SYSTEM <number>  
\*TP ABORTED DURING RECOVERY CYCLE  
\*RECOVERY WILL BE RE-INITIATED WHEN  
\*PROGRAM <number> IS DISABLED AND CLEARED

A user TP abnormally terminated while the recovery cycle was in process. The user must DISABLE and CLEAR the program to re-initiate the recovery cycle.

\*RECOVERY CYCLE WILL BE INVOKED WHEN  
\*PROGRAM <number> IS DISABLED AND CLEARED

This message follows the "PROGRAM TERMINATED" message when a TP has terminated while in DMS II transaction state. The user should DISABLE and CLEAR the program in question to initiate the MCS synchronized recovery cycle.





# APPENDIX F

## DUMP CONTROL INFORMATION

The TCL Processor may be requested to perform a dump of the Input Queue file, the Output Queue file, or the Control file. Certain information must be supplied, either by card or by Supervisory console, initiated by a card-file verification to ensure the file is present. The acceptable card file title is "DUMPCARD." Control information is expected in free-form format, one request per record. A <disk queue index> greater than 10 or card End-of-File causes program End-of-File.

Syntax for the control information is as follows:

```
<dump control information> ::=  
    <disk queue index> ,  
    <start record key> ,  
    <stop record key> ,  
    <access mode> ,  
    <format> ,
```

```
<disk queue index> ::=  
    <input disk queue> /  
    <output disk queue> /  
    <control disk queue> /  
    <end-of-job flag>
```

```
<input disk queue> ::= 0
```

```
<output disk queue> ::= 1
```

```
<control disk queue> ::= 2
```

```
<end-of-job flag> ::= [an integer greater than 10]
```

```
<start record key> ::= <integer>
```

```
<stop record key> ::= <integer>
```

```
<access mode> ::=  
    <serial dump> /  
    <base linked dump> /  
    <station input linked dump>
```

```
<serial dump> ::= 0
```

```
<base linked dump> ::= 1
```

```
<station input linked dump> ::= 2
```

```

<format> ::=
    <EBCDIC and links> /
    <EBCDIC> /
    <EBCDIC, hex and links> /
    <EBCDIC and hex> /
    <formatted EBCDIC and base links> /
    <formatted EBCDIC> /
    <formatted EBCDIC, base links and input message> /
    <formatted EBCDIC and input message>

```

```

<EBCDIC and links> ::= 0

```

```

<EBCDIC> ::= 1

```

```

<EBCDIC, hex and links> ::= 2

```

```

<EBCDIC and hex> ::= 3

```

```

<formatted EBCDIC and base links> ::= 4

```

```

<formatted EBCDIC> ::= 5

```

```

<formatted EBCDIC, base links and input message> ::= 6

```

```

<formatted EBCDIC and input message> ::= 7

```

Example:

0 , 0 , 100 , 0 , 0	[SERIAL ALPHA DUMP WITH LINKS OF INPUT DISK QUEUE FROM KEY 0 TO KEY 100].
1 , 325 , 1025 , 1 , 2	[BASE LINKED ALPHA AND HEX WITH LINKS OF OUTPUT DISK QUEUE FROM BASE KEY 325 TO BASE KEY 1025].

A <serial dump> dumps all records from <start record key> to <stop record key>. The <start record key> must be less than the <stop record key>.

A <base linked dump> dumps records for a particular Input queue (if <input disk queue> is specified) or for a particular station (if <output disk queue> is specified). The dump starts at <start record key> and stops at <stop record key>, both of which must be base records and must correspond to the same station or Input queue depending on the <disk queue index>.

A <station input linked dump> dumps all inputs entered at a particular station. It may be obtained only if <input disk queue> is specified. The dump starts at <start record key> and links back to the <stop record key> both of which must be base records for the station. The <start record key> must be greater than the <stop record key>.

There are six formatting parameters which combine to make the eight <format> types specified in the syntax. These formatting parameters are:

1. EBCDIC causes the EBCDIC equivalent of the specified records to be printed.
2. HEX causes the HEX equivalent of the specified records to be printed. If EBCDIC and HEX are specified, the EBCDIC equivalent is printed over the HEX character-for-character.
3. FORMATTED EBCDIC may be used only if <input disk queue> or <output disk queue> is specified. If <input disk queue> is specified, the dump formats and prints the input message as it was entered at the station. If <output disk queue> is specified, the dump formats and prints the output message as it was issued by the Transaction Processor (TP) or the MCS (i.e., before output formatting takes place).
4. INPUT MESSAGE is always specified with FORMATTED EBCDIC. If <input disk queue> is specified, INPUT MESSAGE has no effect. If <output disk queue> is specified, INPUT MESSAGE dumps the input message associated with each output message that is dumped.
5. LINKS causes the control area preceding each Input or Output Queue file to be broken-out and formatted. Each field is named according to the system field defines listed in Appendix B.
6. BASE LINKS causes the key of the base record for each output message to be dumped. In addition, it causes the date, time, message number, and station number of the base record for each input message to be dumped.



# **APPENDIX G**

## **DOCUMENTATION OF TRANSACTION CONTROL LANGUAGE REPORT LISTING**

This appendix contains a sample Transaction Control language (TCL) run defining a very simple on-line environment of one program, two stations, and one area.

Figure G-1 represents the user input for GEMCOS/UTILITY. As indicated, full table and directory listings are requested through use of the LIST and DIRECTORY CONTROL statements. Listed in figure G-2 are the control file variables for the run. These variables are fully documented in Appendix B under System Control file. System file variables are listed in Figure G-3.

The requested station, program, Input queue, and area tables are illustrated in Figures G-4 thru G-7. Immediately preceding these figures is a key defining the abbreviations used in the table fields. System messages for the sample environment are presented in Figure G-8, followed by the requested system, station, program, Input queue, and area directories in Figures G-9 thru G-14.

GEMCOS/UTILITY ON PACK VERSION 06.000.000 TUESDAY ,01/26/82  
CONTROL = LIST, DIRECTORY, GENERATE;

GLOBAL:

CONTROLPERMANENT = TRUE.  
SYSTEMSPD = COMPUTER/ROOM.  
SYSTEMMONITOR = COMPUTER/ROOM.  
MAXRUNNING = 20.

BEGIN

SYSTEM SAMPLE (1):  
PROGRAM PRGM1 (1) USER:  
TITLE = GEMCOS/SAMPLE/PROG.  
COMMONSIZE = 343.  
MAXCOPIES = 4.  
TIMEOUT = 5.  
INPUTQUEUE SAMPQUE (1):  
MKE = ECHO0(1,0), ECHO1(1,1),  
INQUIRE, UPDATE.  
QUEUEDEPTH = 7.  
TIMELIMIT = 60.  
STATION COMPUTER/ROOM (1):  
NOTITLE = TRUE.  
STATION VIV (2):  
ALTERNATE = COMPUTER/ROOM.  
STATIONBIT (0) = TRUE.  
AREA ALLSTATIONS (1):  
STALIST = /=.

END;

NUMBER OF ERRORS DETECTED = 0.  
SYNTACTIC ITEMS = 147.  
SOURCE INPUT TIME WAS 6 SECONDS.

Figure G-1. GEMCOS/UTILITY User Input

GEMCOS/UTILITY ON PACK VERSION 06.000.000 TUESDAY 01/26/82 0915

SYSTEMSPO = COMPUTER/ROOM  
SYSTEMMONITOR = COMPUTER/ROOM  
SYSTEMROUTINGHEADERKEY =  
SYSTEMMONIDENT = SYSMON?  
SYSTEMSPOIDENT = SYSSPO?  
MULTIPLESTIDENT = MULTIPLE?  
SERVICEMESSAGE = SERVICEMESSAGE?

COMMAX = 19.	TABLE = BOT, TOP.
SCRMAX = 1.	LINIOD = 3, 3.
LINMAX = 2.	STAIAD = 4, 4.
STAMAX = 2.	STATION = 5, 7.
PROMAX = 4.	PROGRAM = 8, 12.
IQUMAX = 2.	INPUTQUEUE = 13, 15.
AREMAX = 0.	AREA = 16, 16.
MKEMAX = 8.	MESSAGEKEYS = 17, 17.
STATIONNAMESMAX = 6.	STATIONNAMES = 19, 19.
PROGRAMNAMESMAX = 6.	PROGRAMNAMES = 20, 20.
INPUTQUEUENAMESMAX = 5.	INPUTQUEUENAMES = 22, 22.
AREANAMESMAX = 3.	AREANAMES = 23, 23.
SYSTEMNAMESMAX = 41.	SYSTEMNAMES = 25, 26.
ALTMAX = 4.	ALTERNATE = 27, 27.
SECURITYMAX = 1.	SECURITYITEMS = 28, 28.
TASKMAX = 4.	OLDTABLELINKS = 0, 0.
MKEMESSAGEMAX = 1.	MKEMESSAGE = 29, 29.
STATIONBITSMAX = 1.	STATIONBITS = 31, 31.
BATCHMAX = -1.	BACKUPBATCHJOBS = 0, 0.
USERMAX = 1.	USERCODE = 33, 33.
USERSTAMAX = 0.	STASECURITYMASK = 35, 35.
MKETBLMAX = 8.	MKETABLE = 36, 44.
USERAVLMAX = 0.	USERAVLMASK = 45, 45.
MKEUSERMAX = 0.	USERSECURITYMASK = 46, 46..

Figure G-2. Control File Variables (Sheet 1 of 2)



MSGIDMAX = 1	MESSAGEID = 47,47.
MSGFMTMAX = 4.	FORMATS = 49,53.
DEVICEMAX = 1.	DEVICENAMES = 54,54.
SYSMAX = 0.	SYSTEMS = 56,56.
SYSMKEMAX = 16.	SYSSTCKMKES = 57,57.
SECURITYITEMMAXBIT = 0.	BACKUP = 59,63.
QUESEGC0] = 3.	DCCODEDATE&TIME = 32,32.
QUESEGC1] = 3.	STADCPATTRIBUTES = 58,58.
QUESEGC2] = 63.	
MSGSDAY = 0.	
DCCODETIME = 0000.	DCCODE DATE = ??/??/ ?
XCLOCK = 6.	TCDATE = 01/25/77
STARTTIME = 6.	START DATE = 01/25/77

Figure G-2. Control File Variables (Sheet 2 of 2)

SYSTEMNAME = SAMPLE (1)	SYSTEMPREFIX =
NUMBER OF PROGRAMS = 2.	
NUMBER OF INPUTQUEUES = 3.	
NUMBER OF MESSAGE KEYS = 4.	
NUMBER OF USERS = 0.	
NUMBER OF STATIONS = 3.	
NUMBER OF AREAS = 1.	
DATA BASE SEQUENCE NUMBER = 1.	

Figure G-3. System File Variables

The following key defines the abbreviations used in Figures G-4 thru G-7.

<u>Abbreviation</u>	<u>Definition</u>
ALT	Has alternates
ARE	Area index
ASN	Assigned to a program
BDC	Capable of Broadcast Select
BIT	Control bit
BOT	Current bottom of queue
BOTBOT	Beginning of queue
CDC	TCL receive address
COP	Number of copies
CSZ	Common size in words
EDI	Has Editor program
EDT	Editor program index
EIO	I/O capability
IMA	Is an alternate
IMQ	Number of input messages today
INP	No-input program
IQU	Queue index
STAIGU	Queue index to which a station is assigned
MAN	Manually disabled
MAS	Routing header master
MKES	Message keys
MUD	Modify program
MPD	Messages today
MSG	Number of output messages today
NAM	TCL name
NLK	Next IQU of linked set of input queues
NUT	No-title station
NQU	Station does not queue output
NUM	TCL-defined number
ORG	Station does not originate messages
PER	Permanent
PRC	Practice mode
PRO	Program index
PRT	A port station
STAPRO	Program index to which a station is assigned
QUE	Current number queued
REN	Multiple inputs
ROU	Requires routing headers
SER	Service Message Station (for Station table)
SER	Has Service program (for Program table)
SEV	Service program index
SHR	Pointer to master copy of program
SIL	Pointer to current input queue record
SPC	Special station
STA	Station index
SYS	Index to station's STASYS parameter
TIT	Title of program

<u>Abbreviation</u>	<u>Definition</u>
TOD	Timeout delayed
TOP	Top of queue
TOT	Total
TSC	TCL send address
TST	Test status
TYP	Type of program

Figures G-4, G-5, and G-6 cannot be reproduced as they appear because of printing constraints.

[illegible]

Figure G-4. Station Tables (1 of 2)

BACKED UP											
BDC	SPC	TSC	CDC	BOTBOT	MAN	MSG	QUE	BOT	TOP	1QU	
					ASN	140	PRC		SIL	PRQ	
0	0	***	***	000001	0	00000	0000	000001	000001	000	
					0	00000	0		000000	000	
1	0	???	???	000002	0	00000	0000	000002	000002	000	
					0	00000	0		000000	000	
1	0	???	???	000003	0	00000	0000	000003	000003	000	
					0	00000	0		000000	000	
0000000000000000001											

Figure G-4. Station Tables (2 of 2)

PROGRAM TABLE								
PRO	NAM	NUM	IQU	COP	PER	REN	INP	MOD
00	CONTROL	099	01	1	1	1	0	0
REMARKS: HANDLES SYSTEM CONTROL MESSAGES AND								
01	PRGM1	001	02	4	0	0	0	0

SHR	TYP	BIT	SER	SEV	EDI	EDT	CSZ	TOD	TIT
00	2	00	0	00	0	00	0	60	CONTROL.
OTHER ASSORTED THINGS.									
01	0	01	0	01	0	01	343	5	GEMCOS/SAMPLE/PROG.

Figure G-5. Program Table

INPUTQUE TABLE								
IQU	NAM	NUM	BOT	BOT NLK	MAN	BACKED UP		
						QUE MPD	BOT	TOP
00	ERROR	000	000001	000	0	0000 00000	000001	000001
01	CONTROL	000	000002		0	0000	000002	000002
02	SAMPQUE	001	000003	002	0	0000 00000	000003	000003

PRONAM	MKES
CONTAINS ALL INVALID MESSAGES	
CONTAINS ALL CONTROL MESSAGES	
CONTROL	? COMPUTE MODE ASSIGN CLOSE CHANGE RUN PURGE QUIT INTERCEPT REFRESH \$
PRGRM1	ECHO0 ECHO1 INQUIRE UPDATE

Figure G-6. Input Queue Table

AREA TABLE				
ARE	NAM	TYPE	NUM	TOT
00	ALLSTATIONS NULL SECURITY	BROADCAST	001	0002

Figure G-7. Area Table

SYSTEM MESSAGES	
MESSAGE INVALIDMESSAGEKEY :	TEXT = INVALID MESSAGE KEY??
MESSAGE SECURITYERROR :	TEXT = INVALID MESSAGE KEY??
MESSAGE TRANSMISSIONERROR :	TEXT = TRANSMISSION ERROR PLEASE RE-SEND ???
MESSAGE UPDATEDISABLED :	TEXT = MESSAGE REJECTED, UPDATE MESSAGES DISABLED
MESSAGE BUSYMESSAGE :	TEXT = BUSY
MESSAGE LOGONOK :	TEXT = #
MESSAGE LOGOFFOK :	TEXT = #

Figure G-8. System Messages

S Y S T E M   D I R E C T O R Y								01/26/82	0915
SYSTEM	NUM	STAS	PROS	IQUS	AREAS	USERS	MKES	DBSN	
SAMPLE	1	3	2	3	1	0	4	0	

Figure G-9. System Directory

S T A T I O N   D I R E C T O R Y				01/26/82	0915
FOR SYSTEM SAMPLE (1)					
NAME	NUMBER	TSC	CDC		
COMPUTER/ROOM	1	???	???		
VIV	2	???	???		

Figure G-10. Station Directory

P R O G R A M   D I R E C T O R Y					01/26/82	0915
FOR SYSTEM SAMPLE (1)						
NAME	NUMBER	TYPE	INPUTQUEUE NUMBER	NAME	EXTERNAL NAME	
CONTROL	99	PROCESS		CONTROL	CONTROL.	
PRGM1	1	USER		SAMPQUE	GEMCOS/SAMPLE/PRG.	

Figure G-11. Program Directory



```

INPUT QUEUE DIRECTORY      01/26/82   0915
FOR SYSTEM USERSYSTEM (8)

NAME          NUMBER      PROGRAM
                NUMBER      NAME

CONTROL
ERROR
SAMPQUE          1          1          PRGM1

```

Figure G-12. Input Queue Directory

```

AREA DIRECTORY (BY STATION)      01/26/82   0915
FOR SYSTEM SAMPLE (1)

STATION      NUMBER      AREAS

COMPUTER/ROOM      1      ALLSTATIONS
VIV                 2      ALLSTATIONS

```

Figure G-13. Area Directory (by Station)

```

AREA DIRECTORY (BY AREA)      01/26/82   0915
FOR SYSTEM SAMPLE (1)

      AREA : ALLSTATIONS(1) BROADCAST

COMPUTER/ROOM
VIV

```

Figure G-14. Area Directory (by Area)

-2241000

- 3 - 17 Stop queue time in hours and minutes
- 8 - 23 Stop queue date in MMDDYY format
- 4 - 26 Number of Input queues, base 1
- 7 - 30 Number of stations, base 1
- 0 - 108 Not defined

# INDEX

- \$ option cards 11-5
- \$SET 11-3, 11-4
- ?SYSTEM EOJ 8-37
- ACCESS CONTROL 1-1, 7-1
- Access Control module 7-1, 7-2, 7-3
- Access control 7-1, 7-2
- ACCESSCONTROL STATEMENT 3-51
- ACCESSCONTROLPROGRAM STATEMENT 3-58
- ADD REQUEST 8-16
- add request 8-16
- ADDRESS STATEMENT 3-80
- administrative message 10-5
- administrative message-switching program 10-5
- ADMINKEY STATEMENT 3-52
- ADMINPROGRAM STATEMENT 3-16
- ALTERNATE STATIONS STATEMENT 3-80
- ALTERNATIVE TO THE BATCH INTERFACE 4-18
- APPLICATION PROGRAMS 4-1
- ARCHIVAL AUDIT TAPE FILE B-29
- Archival Audit tape file B-29
- ARCHIVAL RECOVERY 5-23
- ARCHIVALAUDIT STATEMENT 3-52, 3-71
- AREA (STATION) SECTION 3-105
- ASSIGN INPUT MESSAGE 8-2
- assign input message 8-2, 8-3
- ASSIGNTOPROGRAM STATEMENT 3-81
- AUDIT LEDGER FILE B-29
- Audit Ledger file B-29
- AUDIT STATEMENT 3-72
- AUTORECOVERY 3-53, 5-17
- auxiliary programs 10-1
- AUXILIARY PROGRAMS 10-1
- BASIC COMPONENTS 3-4
- BASIC SYMBOLS 3-3
- BATCH JOB INTERFACE 4-17
- Batch Transaction Processor 1-2
- BATCHIOKEY STATEMENT 3-17
- break variable statement 3-40
- BROADCAST STATEMENT 3-81
- case statement 3-39
- change input message 8-3, 8-4
- CHANGE INPUT MESSAGE 8-3
- change request 8-17
- CHANGE REQUEST 8-17
- CHARACTER SET 3-2
- CHECK 3-112
- CHECKPOINT RECOVERY 5-2

## INDEX (Cont.)

clear request 8-23  
CLEAR request 8-23  
CLEAR REQUEST 8-23  
CLEARRESTART STATEMENT 3-54  
close input message 8-4  
CLOSE INPUT MESSAGE 8-4  
Common area A-1  
COMMON AREA LAYOUT A-1  
COMMONAREA 4-1  
COMMONSIZE STATEMENT 3-60  
compilation options 3-112  
compiler options 11-3  
compute input message 8-4  
COMPUTE INPUT MESSAGE 8-4  
COMPUTE STATEMENT 3-82  
CONTENTS OF RELEASE TAPE 11-1  
CONTINUOUS PROCESSING 9-1  
continuous processing 9-1  
CONTINUOUSLOGON STATEMENT 3-82  
CONTINUOUSPROCESSING STATEMENT 3-17  
CONTINUOUSPROCESSING 9-1  
CONTROL SECTION 3-7  
CONTROLBIT STATEMENT 3-60  
CONTROLLEDPAGING STATEMENT 3-18  
CONTROLPERMANENT STATEMENT 3-18  
CONVERSATIONAL INTERFACE 4-22  
CONVERSATIONAL STATEMENT 3-83  
CONVERSATIONSIZE STATEMENT 3-60  
DATA BASE RELATIONSHIPS 2-4  
Data Communications Processor (DCP) 12-1  
data communications Utility B-1  
DATA COMMUNICATIONS UTILITY FILES B-1  
DATACOM SYSTEM B-1  
DATAQUEUES 4-12  
DCP 12-1  
DECLAREDPRIORITY STATEMENT 3-61  
DEFINE INVOCATION 3-13  
DEFINE SECTION 3-12  
DESK DESCRIPTION 3-6  
DEVICE SECTION 3-103  
DIALIN STATEMENT 3-83  
DIRECTORY OLD 3-8  
DIRECTORY 3-8  
DISABLE REQUEST 8-23  
disable request 8-23  
DOCUMENTATION OF TRANSACTION CONTROL LANGUAGE REPORT G-1  
dollar sign (\$) 3-112  
dollar sign input message 8-9

## INDEX (Cont.)

dump control information F-1  
DUMP CONTROL INFORMATION F-1  
DUMP OLD 3-8  
DUMP REQUEST 8-25  
DUMP 3-8  
DYNAMIC VOLUME CONTROL 1-3  
EDITOR PROGRAM INTERFACE 4-12  
Editor program 3-61, 4-1, 4-12  
EDITOR STATEMENT 3-61  
EDITOREVENT 4-1  
ENABLE REQUEST 8-25  
enable request 8-25  
events 4-1  
FAMILY STATEMENT 3-61  
FILE GENERATION 3-110  
FIXEDLENGTHMKE STATEMENT 3-83  
FLUSHOUTPUT STATEMENT 3-84  
FLUSHRECOVERY STATEMENT 3-54  
FLUSHRECOVERY 5-1  
FORMAT AND FUNCTION SECTION 3-27  
FORMATGENERATOR STATEMENT 3-19  
FORMATSIN STATEMENT 3-104  
FORMATSOUT STATEMENT 3-104  
FORMATTEDBLOCKEDOUTPUT STATEMENT 3-84  
FORMATTER 4-12  
FORMATTING 1-2, 6-1  
formatting 6-1  
forms request 6-2  
FORMS REQUEST 6-2  
FORMSCOMPOSE STATEMENT 3-84  
GENERATE 3-9  
GETMESSAGEPROCEDURE 4-7  
GLOBAL SECTION 3-14  
HOST STATEMENT 3-54, 3-62  
IDENTIFY STATEMENT 3-85  
INACTIVETIMEOUT 3-63  
INDIVIDUALID STATEMENT 3-86  
INPUT FORMATTING 6-1  
input/output queue record size statement 3-19  
INPUTQUEUE SECTION 3-70  
INPUTUSERHOOK procedure 3-74, 4-20  
intercept input message 8-9, 8-10  
INTERCEPT INPUT MESSAGE 8-9  
INTERCEPT STATEMENT 3-86  
INTERNAL MCS ACCESS CONTROL MECHANISM 7-1  
INTERROGATE REQUEST 8-27  
interrogate request 8-27  
LIG ERROR 3-9

## INDEX (Cont.)

line count statement 3-35  
LINE STATEMENT 3-86  
LINEANALYZER STATEMENT 3-87  
LIST OLD 3-9  
LIST 3-9, 3-113  
LISTING1 G-1  
LOG INPUT 3-9  
LOG OUTPUT 3-9  
LOG 3-9  
LOGICALACK STATEMENT 3-87  
LOGOFFMKE STATEMENT 3-20  
LOGONMKE STATEMENT 3-20  
MASTERCOMPUTE STATEMENT 3-21  
MASTEREVENT 4-1  
MAXCOPIES STATEMENT 3-63  
MAXNEWACCESSKEYS STATEMENT 3-21  
MAXNEWFORMATS STATEMENT 3-21  
MAXNEWMKES STATEMENT 3-21  
MAXPATCHJOBS STATEMENT 3-21  
MAXRUNNING STATEMENT 3-22  
MAXUSERS STATEMENT 3-88  
MAYNOTBEASSIGNED STATEMENT 3-64  
MAYNOTBEINTERCEPTED STATEMENT 3-88  
MCS ERROR MESSAGES E-1  
MCS STATION MONITOR D-1  
MCS/NDL INTERFACE 12-1  
MEMORYLIMIT STATEMENT 3-73  
MERGE 3-113  
Message Recall program 10-1  
MESSAGE RECALL 10-1  
MESSAGE SECTION 3-108  
MINCOPIES STATEMENT 3-64  
mke position statement 3-102  
MKE STATEMENT 3-73  
mode input message 8-10  
MODE INPUT MESSAGE 8-10  
MODIFY STATEMENT 3-64  
MONITOR D-1  
monitor D-1  
move request 8-30  
MOVE REQUEST 8-30  
MULTIPLEINPUTS STATEMENT 3-65  
MULTIPLESTIDENT STATEMENT 3-22  
MYSUSE STATEMENT 3-88  
NDL 12-1  
NETWORK CONTROL 1-3  
Network Control 8-1  
NETWORK MANAGEMENT AND CONTROL 8-1  
NEW 3-113

## INDEX (Cont.)

Newtork management and control 8-1  
NO RECOVERY 5-1  
NODUPPLICATES STATEMENT 3-89  
NOINPUT STATEMENT 3-74  
NOLINE STATEMENT 3-89  
NONFORMATTEDMKES STATEMENT 3-105  
NOQUEUE STATEMENT 3-89  
NOTITLE STATEMENT 3-90  
OBJECT JOB INTERFACE 4-17  
on-line Datacom dump 9-1  
On-Line Datacom dumps 9-1  
ON-LINE DATACOM DUMPS 9-1  
ONEOUTPUTPERBACKUP STATEMENT 3-90  
OPTION CONTROL CARDS 3-111  
OUTPUT FORMATTING 6-2  
OUTPUT ROUTING 3-107  
PAGE 3-113  
PAGED FORMATS 6-3  
PAGING 1-3, 6-3  
paging 6-3  
PAGINGPERMANENT STATEMENT 3-23  
PATCH 3-113  
PERMANENT STATEMENT 3-65  
POP 3-113  
PORT 4-1  
Port-type program 4-7  
PORTPROGRAMINTERFACE 4-7  
PROCESS PROGRAM INTERFACE 4-5  
Process program 4-5  
PROCESS 4-1  
PRODUCT GENERATION 11-3  
PRODUCT MAINTENANCE 11-5  
PROGRAM GENERATION AND MAINTENANCE 11-1  
PROGRAM SECTION 3-55  
PROGRAMACK STATEMENT 3-93  
QUEUEBLOCKSIZE STATEMENT 3-23  
QUEUEBLOCKSIZE 9-1  
QUEUEDEPTH STATEMENT 3-75  
QUEUERECORDSIZE STATEMENT 3-19  
quit input message 8-11  
QUIT INPUT MESSAGE 8-11  
Recall program 10-1  
RECALLKEY STATEMENT 3-55  
RECALLPROGRAM STATEMENT 3-23  
recover request 8-31  
RECOVER REQUEST 8-31  
RECOVERY OF A NON-DMS TRANSACTION 5-25  
RECOVERY STATEMENT 3-75

## INDEX (Cont.)

RECOVERY 1-3, 5-1  
REFRESH OUTPUT MESSAGE 8-11  
refresh output message 8-11  
REGENERATE 3-10  
release request 8-31  
RELEASE REQUEST 8-31  
REMARKS STATEMENT 3-66, 3-93  
RESET 3-113  
Restart program 5-6  
RESTARTPROGRAM STATEMENT 3-66  
RETURNMESSAGES STATEMENT 3-66  
ROUTEHEADER STATEMENT 3-93  
ROUTING 1-2  
RUN PROGRAM REQUEST 8-31  
run program request 8-31, 8-32  
SENDMESSAGE 4-7, 8-1  
SENT MESSAGEPROCEDURE 4-7  
SEQ 3-113  
SEQERR 3-113  
SEQUENCECONTROL STATEMENT 3-10  
SERVICE PROGRAM INTERFACE 4-14  
Service program 4-1, 4-14  
SERVICE STATEMENT 3-67  
SERVICEEVENT 4-1  
SERVICEMESSAGE IDENT STATEMENT 3-24  
SERVICEMESSAGE STATEMENT 3-94  
SET 3-113  
SETMCSREQUESTSET STATEMENT 3-94  
SIGNON STATEMENT 3-95  
SINGLE 3-114  
SM (Send Messages) 8-1  
SPECIAL STATEMENT 3-95  
STALIST STATEMENT 3-105, 3-106  
STAMASTER STATEMENT 3-96  
STASYSTEM STATEMENT 3-97  
station section 3-6  
STATION SECTION 3-76  
STATION-TO-STATION ADMINISTRATIVE MESSAGES 10-5  
STATIONBIT STATEMENT 3-95  
status request 8-32  
STATUS REQUEST 8-32  
SUBSPACES STATEMENT 3-67  
SUBSYSTEM STATEMENT 3-55, 3-67  
SUBSYSTEMS STATEMENTS 3-24  
SUBTRACT REQUEST 8-36  
subtract request 8-36

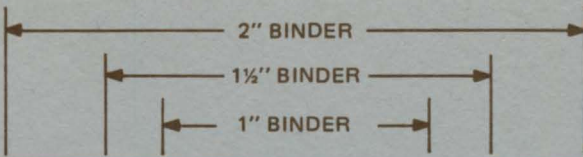
## INDEX (Cont.)

SWITCH INPUT MESSAGE 8-12  
switch input message 8-12  
switch station define 3-79  
SYNCHRONIZED RECOVERY OF BATCH JOBS 5-25  
SYNCHRONIZED RECOVERY OF TP-TO-TP TRANSACTIONS 5-26  
SYNCHRONIZED RECOVERY 5-2  
SYNTAX 3-114  
SYSPC 3-26  
SYSTEM CAPABILITIES 1-1  
System Control file B-3  
SYSTEM CONTROL FILE B-3  
system control messages 8-1, 8-2, 8-12, 8-13  
SYSTEM DEFINED INPUT MESSAGE 8-2  
system defined input message 8-2  
System Defined Input Message 8-2  
system disk queue files B-7  
SYSTEM DISK QUEUE FILES B-7  
System error messages 8-46  
SYSTEM ERROR MESSAGES 8-46  
SYSTEM FORMAT FILES B-12  
SYSTEM LOG FILES B-23  
SYSTEM OVERVIEW 2-1  
system request 8-36  
SYSTEM REQUEST 8-36  
SYSTEM SECTION 3-50  
SYSTEM SERVICE MESSAGES 8-51  
System Service messages 8-51, 8-52  
SYSTEMCONTROLMESSAGES 8-12  
SYSTEMIOKEY STATEMENT 3-26  
SYSTEMMONIDENT STATEMENT 3-25  
SYSTEMMONITOR STATEMENT 3-25  
SYSTEMNETWORKCONTROL STATEMENT 3-25, 3-98  
SYSTEMNETWORKCONTROLIDENT STATEMENT 3-26  
SYSTEMPREFIX STATEMENT 3-27  
SYSTEMSPCIDENT 3-26  
SYSTEMSPOSTATION 3-98  
TABLE REQUEST 8-38  
table request 8-38  
TCL-DIRECTING OPTIONS 3-112  
TEST STATEMENT 3-99  
TESTPROGRAM STATEMENT 3-68  
THE CONTROL WORDS ARRAY C-1  
TIMELIMIT STATEMENT 3-75  
TIMEOUT STATEMENT 3-68  
TITLE STATEMENT 3-69  
TRANSACTION CONTROL LANGUAGE 1-1, 3-1  
TRANSACTION PROCESSING SYSTEM (TPS) RECOVERY 5-28



## INDEX (Cont.)

TRANSACTION PROCESSORS 4-1  
TRANSACTIONMODE STATEMENT 3-99  
TRANSMISSIONERRORMESSAGES 3-69  
UPDATE PAGING 6-4  
UPDATE REQUEST 8-41  
UPDATEFMT 3-12  
USAGE AND EXAMPLES 11-7  
USER PROGRAM INTERFACE 4-1  
USER ROUTING PROCEDURE 4-20  
USER 4-1  
USER-SUPPLIED ACCESS CONTROL MODULE 7-1  
user-supplied Access Control module 7-2, 7-3  
USERCONTROL STATEMENT 3-101  
USEREVENT 4-1  
USEREVENTARRAY 4-12, 4-14  
USERIDMASK STATEMENT 3-101  
USERIDPOSITION 3-102  
VALIDUSERS STATEMENT 3-101  
VARIABLEMKEPOSITION STATEMENT 3-102  
VARIABLEUSERID POSITION 3-102  
VOID 3-114  
VOIDT 3-114  
WAITFORAUDIT PROGRAM EXAMPLE 5-13  
WAITFORAUDIT STATEMENT 3-76  
WAITFORAUDIT 4-8, 5-3, 5-5  
WAITFORAUDITEVENT 4-8  
zip request 8-45  
ZIP REQUEST 8-45



**B 5000/B 6000/B 7000 Series**  
**Generalized Message Control System (GEMCOS)**  
**USER'S/REFERENCE MANUAL**

1096567

Printed in U.S.A.