


Burroughs 

**B 1700/B 1800
Generalized Message
Control System
(GEMCOS)**

USER'S MANUAL

PRICED ITEM

Burroughs 

**B 1700/B 1800
Generalized Message
Control System
(GEMCOS)**

USER'S MANUAL

Copyright © 1980 Burroughs Corporation, Detroit, Michigan 48232

PRICED ITEM

“The names used in this publication are not of individuals living or otherwise. Any similarity or likeness of the names used in this publication with the names of any individuals, living or otherwise, is purely coincidental and not intentional.”

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	xi
1	SYSTEM OVERVIEW	1-1
	General	1-1
	MCS Program	1-1
	Transaction Control Language Compiler	1-3
	MCSTIC	1-3
	MCSFORMATS	1-3
	Auxiliary Programs	1-3
	MCSSIM	1-3
	MCSFIX	1-3
	MCSRECALL	1-3
2	CAPABILITIES	2-1
	Application-Program Interface	2-1
	Transaction-Based Routing	2-3
	Access Control	2-3
	Access Security	2-3
	Process Security	2-4
	Network Administration	2-4
	Error Handling	2-6
	Network Restoration	2-6
	Message Recovery	2-7
	Audit	2-7
	Recovery	2-7
	Controlled Shutdown	2-7
	Mergeable External Source Statements	2-8
	Debugging Aids	2-9
	Data Dump	2-9
	Monitor Trace	2-9
	Screen Wraparound	2-9
	Supervisory MCS	2-10
	Remote Program Execution	2-10
	Formatting	2-10
3	TRANSACTION CONTROL LANGUAGE	3-1
	General Discussion of Transaction Control Language (TCL)	3-1
	Syntax Conventions	3-1
	Metalinguistic Symbols	3-1
	Metalinguistic Formulae	3-2
	Character Set	3-3
	Basic Symbols	3-4
	Basic Components	3-5
	Identifiers	3-6
	Generalized Identifiers	3-7
	Strings	3-9
	Logical Values	3-10

Table of Contents (cont)

Section	Title	Page
3 (cont)	Deck Description	3-11
	CONTROL Statement	3-13
	MCSTIC FILE NAME Statement	3-18
	FORMAT FILE NAME Statement	3-19
	GLOBAL Section	3-20
	CHANGE REQUESTS Statement	3-22
	DATA DUMP Statement	3-23
	MESSAGE BROADCAST Statement	3-24
	MESSAGE RECALL Statement	3-25
	PROGRAM BOJ EOJ Statement	3-26
	MONITOR TRACE Statement	3-27
	STATUS REPORTS Statement	3-28
	SYSTEM HALT Statement	3-29
	COMPILE OPTIONS Statement	3-30
	OBJECT-CODE FILE NAME Statement	3-31
	SOURCE-CODE FILE NAME Statement	3-32
	NAME-STACK ENTRIES Statement	3-33
	VALUE-STACK BITS Statement	3-34
	CONVERSATIONLIMIT Statement	3-35
	NCC OK RESPONSE Statement	3-36
	SIGNAL CHARACTER Statement	3-37
	AUDIT RECORD SIZE Statement	3-38
	AUDIT PAGE SIZE Statement	3-39
	AUDIT FILE PACK ID Statement	3-40
	CHECKPOINT INTERVAL Statement	3-41
	MAXIMUM TEXT SIZE Statement	3-42
	QUEUE DEPTH Statement	3-43
	QUEUE BUFFERS Statement	3-44
	QUEUE NAME Statement	3-45
	SIMULATION Statement	3-46
	MONITOR TRACE ON Statement	3-47
	FORMAT AND FUNCTION Statement List	3-48
	Function Declaration	3-50
	Format Declaration	3-53
	Formatting Errors	3-60
	Basic GEMCOS Formatting Pragmatics	3-66
	RECALL PROGRAM Statement	3-72
	CONTROL STATIONS Statement	3-73
	DEFINITION Section	3-74
	ACCESS CONTROL Statement	3-75
	PROGRAM Section	3-77
	Assignment Programs	3-78
	Utility Programs	3-79
	User Programs	3-80
	INTERFACE Statement	3-82
	NONPARTICIPATION	3-82
	PARTICIPATION	3-84
	MCS	3-85
	TRANCODE Statement	3-88
	PROGRAM TITLE Statement	3-89
	RESIDENCE Statement	3-90

Table of Contents (cont)

Section	Title	Page
3 (cont)	COMMON SIZE Statement	3-91
	EXECUTE Statement	3-92
	RECOVERY Statement	3-94
	DATA BASE NAME Statement	3-95
	AUDIT TRANSACTIONS Statement	3-96
	AUDIT ASSIGNMENT Statement	3-97
	AUDIT OUTPUT Statement	3-98
	RESTART PROGRAM Statement	3-99
	MAXIMUM COPIES Statement	3-100
	OPEN MESSAGE Statement	3-101
	ATTACH MESSAGE Statement	3-102
	DETACH MESSAGE Statement	3-103
	CONVERSATIONSIZE Statement	3-104
	MAXASSIGNERS Statement	3-105
	TRANSACTION CODE POSITION Statement	3-106
	AP300STATUS Statement	3-107
	STATION Section	3-108
	SIGN-ON Statement	3-110
	SCREEN SIZE Statement	3-111
	TRANSACTION CODE POSITION Statement	3-112
	VALID ACCESS KEYS Statement	3-113
	TRANSACTION MODE Statement	3-114
	CONTINUOUS LOG-ON Statement	3-115
	CONVERSATIONAL Statement	3-116
	TRANCODE Statement	3-117
	TYPE Statement	3-118
	DEVICE Section	3-119
	STATION LIST Statement	3-121
	INPUT FORMATS Statement	3-122
	OUTPUT FORMATS Statement	3-123
	MESS CODE Section	3-124
	Static Declarations	3-125
	Dynamic Declarations	3-126
	Procedure Define List	3-127
	MESS Procedures	3-129
	SET SIZES	3-129
	SET VALUES	3-130
	MESSAGE FROM STATION	3-131
	MESSAGE FROM PROGRAM	3-131
	MAINTENANCE	3-132
	AUDIT	3-133
	ERROR HANDLER	3-133
	CLOSE FILES	3-133
	HANDLE RECALL	3-134
	INITIATE RESTORE	3-134
	RESTORE PROGRAM	3-134
	OPEN ACTION	3-135
	CLOSE ACTION	3-135
	Common-area Header	3-136
	MONITOR	3-144
	Network Control Commands	3-145

Table of Contents (cont)

Section	Title	Page
3 (cont)	Security Control Commands	3-147
	SIGN ON (SGN)	3-147
	SIGN OFF (BYE)	3-148
	ENABLE USER (EUS)	3-149
	DISABLE USER (DUS)	3-150
	Program Control Commands	3-151
	EXECUTE PROGRAM (EX)	3-151
	HALT APPLICATION PROGRAM (HAP)	3-152
	MCS Control Commands	3-153
	HALT SYSTEM (HLT)	3-154
	Message Control Commands	3-155
	BROADCAST (BRC)	3-155
	POP QUEUE (PQ)	3-156
	Report Commands	3-157
	REPORT DATA DUMP (RDM)	3-157
	REPORT FILE STATUS (RFS)	3-158
	REPORT PROGRAM STATUS (RPS)	3-159
	REPORT PROGRAM COUNTERS (RPC)	3-160
	REPORT STATION COUNTERS (RSC)	3-161
	REPORT STATION STATUS (RSS)	3-162
	Change Commands	3-163
	CHANGE MONITOR FLAG (CMF)	3-163
	CHANGE STATION ADDRESS (CSA)	3-164
	CHANGE STATION DIAGNOSTIC (CSD)	3-165
	CHANGE STATION FREQUENCY (CSF)	3-166
	CHANGE STATION MAXIMUM RETRY (CSM)	3-167
	CHANGE STATION QUEUE (CSQ)	3-168
	CHANGE STATION READY (CSR)	3-169
	CHANGE STATION TRANSMISSION NUMBER (CST)	3-170
	FORMATUPDATE (UPD) Command	3-171
	Audit & Recovery Commands	3-172
	RECOVER DATA BASE (REC)	3-174
	RESET BUSY STATUS (RBS) Command	3-175
4	MESSAGE ROUTING	4-1
	General	4-1
5	ACCESS CONTROL	5-1
	General	5-1
6	MESSAGE FORMATTING	6-1
	General	6-1
	GEMCDS Editing Phrases	6-1
	Output Formatting Example	6-3
	Input Formatting Example	6-11

Table of Contents (cont)

Section	Title	Page
7	SYSTEM OPERATION	7-1
	Compiling with MCSTCL	7-1
	Executing an MCS	7-1
	Console or Card Reader Input	7-1
	Recovery Procedure	7-2
8	AUXILIARY PROGRAMS	8-1
	MCSSIM	8-1
	Example Simulation Card Deck	8-3
	MCSFIX	8-3
	MCSRECALL	8-6
9	RECOVERY	9-1
	General	9-1
	No Recovery	9-1
	Types of Recovery	9-2
	Queue Restoration Recovery	9-2
	Nonsynchronized and Synchronized Data	
	Base Recovery	9-3
	Remote File Opens	9-4
	Transaction Processing	9-6
	End-Of-Job	9-6
	Program Abort	9-6
	Recovery Processing	9-7
	Recovery After System Failure	9-7
	Data Base Recovery (Nonsynchronized)	9-7
	Synchronized Recovery	9-8
	Recovery-Related Conventions	9-9
	Restart Program	9-11
	Recovery Cycle	9-11
	Archival Recovery	9-13
	Conclusion	9-15
10	STATION TYPES	10-1
	General	10-1
	MT600	10-1
	Processing Input From The MTS	10-2
	Processing Output To The MTS	10-2
	Modular Terminal System Message Types	10-2
11	CONVERSATIONAL FEATURE	11-1
	General	11-1
	TCL Specifications	11-1
	CONVERSATIONLIMIT Statement	11-1
	CONVERSATIONSIZE Statement	11-2
	CONVERSATIONAL Statement	11-2
	Procedures for Conversational Programs	11-3

Table of Contents (cont)

Section	Title	Page
11 (cont)	Recovery of Conversational Programs	11-7
	Summary	11-7
APPENDIX A	SUMMARY OF TCL SYNTAX	A-1
APPENDIX B	SUMMARY OF NETWORK CONTROL COMMANDS	B-1
APPENDIX C	SYSTEM REQUIREMENTS	C-1
APPENDIX D	MCS OUTPUT MESSAGES	D-1
APPENDIX E	SUMMARY OF FILES	E-1
APPENDIX F	COBOL74	F-1
APPENDIX G	TCL SIZE LIMITATIONS	G-1
INDEX	one

List of Illustrations

Figure	Title	Page
1-1	System Structure	1-2
3-1	Files Created by TCL Compiler When CONTROL = LIST	3-15
3-2	Files Created by TCL Compiler When CONTROL = GENERATE, LIST, or COMPILE	3-16
3-3	Files Created by TCL Compiler When CONTROL = REGENERATE or LIST	3-17
3-4	Example Set 1 - Formatting Specifications Applied to Input and Output Messages	3-61
3-5	Example Set 2 - Formatting Specifications Applied to Input and Output Messages	3-62
3-6	Example Set 3 - Formatting Specifications Applied to Input and Output Messages	3-63
3-7	Example Set 4 - Formatting Specifications Applied to Input and Output Messages	3-64
3-8	Example Set 5 - Formatting Specifications Applied to Input and Output Messages	3-65
3-9	Initial Contents of Terminal and Program Message Buffers	3-67
3-10	Contents of Terminal/Message Buffers After Move Caused by A3 <item phrase>	3-67
3-11	Contents of Terminal/Message Buffers After Move Caused by I4 <item phrase>	3-68
3-12	Contents of Initialized Buffers	3-68
3-13	Buffer/Pointer Update After Applying Specification A2	3-69
3-14	Buffer/Pointer Update After Applying Specification A4	3-69

List of Illustrations (cont)

Figure	Title	Page
3-15	Buffer/Pointer Update After Applying Specification A1	3-70
3-16	Buffer/Pointer Update After Applying Specification X1	3-70
3-17	Buffer/Pointer Update After Applying Specification Q3	3-71
3-18	Buffer/Pointer Updates After Applying Specification A1 and Sending the Terminal Message Buffer Contents	3-71
3-19	NONPARTICIPATION Interface	3-82
3-20	PARTICIPATION Interface	3-84
3-21	MCS Interface	3-85
6-1	Sample Formatted Output Data Display	6-5
6-2	Example Formatted Output Data Display for Redefined Forms Function	6-7
6-3	Example Formatted Output for TD830 Terminal	6-11
6-4	Example Input Form for TD700 Terminal	6-12
6-5	Example Input Form for TD830 Terminal	6-14
6-6	Example Filled-In Input Form	6-17
8-1	Example Simulation Card Deck	8-3
9-1	Recommended DMS II Restart Data Set Definition	9-4
9-2	Recommended Remote File Record Definition	9-5
9-3	User Program Abnormal Termination Recovery Cycle	9-12
9-4	MCS - Archival Recovery Cycle	9-14
11-1	Recommended Remote File Record and Working Storage Definition	11-4
A-1	Syntax of <MCSTCL deck description>	A-3
A-2	Syntax of <GLOBAL section>	A-4
A-3	Syntax of <DEFINITION section>	A-5
A-4	Syntax of <ACCESS CONTROL statement>	A-5
A-5	Syntax of <PROGRAM section>	A-6
A-6	Syntax of <STATION section>	A-7
A-7	Syntax of <DEVICE section>	A-7
A-8	Syntax of <MESS CODE section>	A-8
A-9	Syntax of <FUNCTION and FORMAT statement>	A-9
A-10	Syntax of <function declaration>	A-9
A-11	Syntax of <format declaration>	A-10
A-12	Syntax of <local declaration part>	A-10
A-13	Syntax of <editing specification>	A-11
A-14	Syntax of <editing string>	A-11
A-15	Syntax of <item phrase>	A-12
A-16	Syntax of <field width>	A-12
A-17	Syntax of <variable identifier>	A-13
A-18	Syntax of <internal size>	A-13
A-19	Syntax of <hex unit string>	A-13
A-20	Syntax of <hexadecimal string>	A-14

List of Tables

Table	Title	Page
3-1	Common-area Header Fields Containing Valid Information by MCSTYPE	3-143
9-1	Recovery Control Messages	9-15
E-1	Summary of GEMCOS System Files	E-1

INTRODUCTION

Telecommunications devices are being used widely to give more people direct access to computer services and to make computer systems more responsive to the needs of the people who use them. However, along with the benefits of telecommunications come increased programming costs required to handle the special problems of the on-line environment. Burroughs B 1800/B 1700 Generalized Message Control System (GEMCOS) is designed to help reduce these costs by providing services which can enhance most on-line application programs.

This manual explains the system with high-level descriptions of components and capabilities and detailed specifications of how to use GEMCOS.

GEMCOS is available in three versions in order to accommodate any level of operating complexity. These versions are the Basic Version, the Advanced Version and the Total Version. The major capabilities of each of these versions are:

- a. Basic Version GEMCOS:
 - 1) Transaction Control Language (TCL).
 - 2) Access control.
 - 3) Message routing.
 - 4) Message auditing.
 - 5) Network control.
 - 6) Message recovery.
 - 7) Nonparticipating MCS.
- b. Advanced Version GEMCOS:
 - 1) All Basic Version capabilities.
 - 2) Message formatting.
- c. Total Version GEMCOS:
 - 1) All Advanced Version capabilities.
 - 2) Data base and synchronized recovery.

The style identification numbers for the GEMCOS product are B1800/B1700 GPB, GPA, AND GPT when purchased with the UPL Compiler, or B1800/B1700 MCB, MCA, AND MCT when purchased without it.

The material in this manual is supplemented by those portions of the following documents which pertain to Message Control System or Network Controller interfaces:

- a. Burroughs B 1700 Systems Network Definition Language Reference Manual, form 1073715.
- b. Burroughs B 1700 Systems User Programming Language Reference Manual, form 1067170.

- c. Burroughs B 1800/B 1700 Systems System Software Operational Guide, form 1068731.
- d. Burroughs B 1700 Systems COBOL Reference Manual, form 1057197.
- e. Burroughs B 1700 Systems Report Program Generator Reference Manual, form 1057189.
- f. Burroughs B 1700 Systems Data Communications Information Manual, form 1089992.
- g. Burroughs B 1800/B 1700 Generalized Message Control System (GEMCOS) Formatting Guide, form 1106531.
- h. Burroughs B 1800/B 1700 Generalized Message Control System (GEMCOS) Capabilities Manual, form 1106572.

SECTION 1
SYSTEM OVERVIEW

GENERAL.

GEMCOS is a system of programs and files whose purpose is to create and support a Message Control System (MCS).

MCS PROGRAM.

An MCS manages the flow of messages between the Network Controller (NC) and the application system (see figure 1-1). The MCS program created by GEMCOS is designed to:

- a. Direct messages among telecommunications stations and application programs according to user-defined routes.
- b. Keep unauthorized persons from using telecommunications stations to gain access to the application.
- c. Require authorized personnel to perform certain functions.
- d. Prevent data communications errors from affecting application programs or unduly impeding operations in the rest of the data communications network.
- e. Gather statistics about data communications errors to aid in diagnosing hardware problems.
- f. Adapt dynamically to changing conditions in the network.
- g. Log messages and return them to application programs on demand.
- h. Inform the network controller of the network status before a system failure.
- i. Provide an orderly network shutdown, accounting for all messages in process.
- j. Segment messages (if necessary) to fit the buffer of destination stations.
- k. Provide remote stations the ability to execute programs.
- l. Allow remote stations to switch between MCS programs.
- m. Furnish useful information to application programs.
- n. Format messages and provide a screen request function (advanced version only).
- o. Allow off-line testing of the MCS.

p. Provide debugging aids.

q. Provide audit and recovery capability at the program or data base level.

The MCS is generative, so not all of these capabilities need to be present in any specific version of the MCS. The generative feature also allows for inclusion of user-written code in the MCS to supplement or supplant its standard functions.

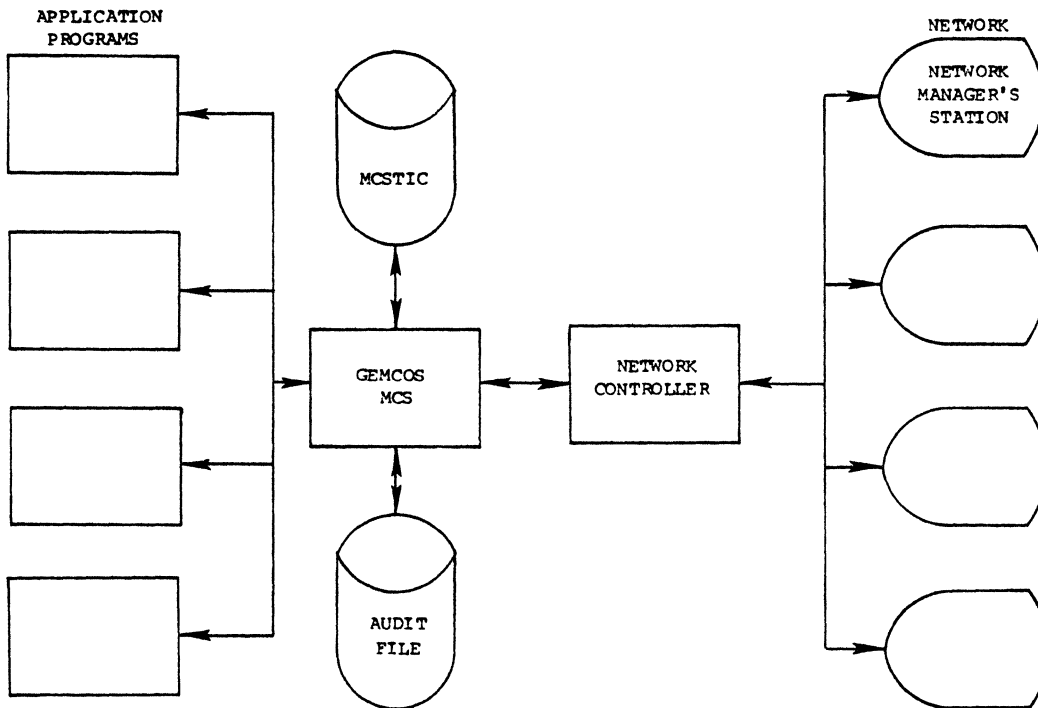


Figure 1-1. System Structure

TRANSACTION CONTROL LANGUAGE COMPILER.

The Transaction Control Language (TCL) is the means by which the user describes the data communications environment and requirements. The user specifies such information as message routing criteria, access control requirements, and MCS options using TCL.

TCL is a free-form structured language utilizing key words to describe the environment and requirements of the data communications user. Compiling TCL generates a set of tables describing the user data communications system and, if requested, MCS code. The compiler (MCSTCL) produces an optional hard-copy listing of the user's data communication system. It also provides extensive syntax checking to ensure that the MCS is properly defined.

The TCL for the B 1800/B 1700 GEMCOS is similar to the TCL of CMS GEMCOS and B 7000/B 6000 GEMCOS.

MCSIIC.

The Table Information Control file, MCSTIC, is used by the MCS to store some of its important variables and parameters. By storing them in a disk file, they are preserved from one execution of the MCS to the next and are protected in case the MCS is unconventionally terminated. Moreover, should system relationships change, the TCL compiler rewrites the MCSTIC file.

MCSFORMATS.

In the advanced and total versions of GEMCOS, all functions and formats created by the TCL compiler are stored in a separate disk file called MCSFORMATS.

AUXILIARY PROGRAMS.

In addition to the previously mentioned programs, the system contains three utility programs: MCSSIM, MCSFIX, and MCSRECALL.

MCSSIM.

The program MCSSIM is used to test the MCS and user-written code off-line. Simulated input messages are presented to MCSSIM using a card reader. MCSSIM forwards them to the MCS as if they had come from the Network Controller. Outputs are printed on a system printer.

MCSFIX.

This program patches and/or lists source-code files which are on disk in the User Programming Language (UPL) format.

MCSRECALL.

This program allows the user at a station to recall audited input or output messages. These messages can be sent either to the station or to the system printer.

SECTION 2

CAPABILITIES

APPLICATION-PROGRAM INTERFACE.

The MCS communicates with Application Programs through the remote file interface, but it provides some optional extensions to the standard interface. Application Programs may be written in any language that supports remote or data communications files.

A standard remote file provides a link between an application program and one or more data communications stations, which were declared as belonging to that file in the FAMILY statement of NDL. Messages are exchanged in the record area of the file. In COBOL, an actual key can be used to exchange information about the message, namely, message type, message length, and source or destination station. Stations are identified by a number which is relative to the order in which the stations were assigned to the file in the FAMILY statement.

GEMCOS offers a Common-area header option which overrides portions of the standard interface. Common area is a header which appears in the record area of a file in front of the message. Common area contains the following fields:

- a. Logical Station Number (LSN) - identifies the source or destination station. The LSN is a number which is relative to the order in which stations are defined in the STATION section of NDL. This identifier is not necessarily the same as that used in the actual COBOL key.

NOTE

The LSN of a given station changes if stations are presented in a different order when the NDL is recompiled.

- b. Message type - distinguishes between normal messages and messages used for special purposes, such as recovery, FILE OPEN notification, FILE ATTACH notification, and FILE DETACH notification.
- c. Sequence number - assigned by the MCS, primarily to aid in recovery.
- d. NDL time - contains the time at which the Network Controller sends the message to the MCS.
- e. Message text size - indicates the length (in bytes) of an incoming message, not including the Common-area header.
- f. Terminal type - identifies the kind of terminal or device from which the message originated.
- g. Message identifier - allows Application Programs to specify which format, if any, is to be applied during output (advanced version only).

- h. **Trancode (transaction code) index 1 - assists the application program to determine which transaction was received.**
- i. **Trancode index 2 - further assists the application program to determine which transaction was received.**
- j. **Format error indicator - notifies the application program that at least one field of an input message is in error (advanced version only).**
- k. **Input address - address in the audit file of this message.**
- l. **Retry count - the number of times this message was submitted to an Application Program.**
- m. **Recovery status - indicates if this message is a normal message or a recovered message.**
- n. **Output address - address in the audit file of the corresponding output message.**
- o. **User area - allows the user to reserve space in the Common-area header for user-defined fields.**

When the Common-area header is used, the MCS attaches the Common-area header to each message sent to an Application Program and removes it

from each message received from an application program, so that the Common-area header does not accompany the message on the Network Controller side of the MCS.

The actual key in COBOL should only be used for specifying the text size of an outgoing message.

When an Application Program receives a message from the MCS, the LSN field in the Common-area header is set to the originating station. As long as the LSN remains unmodified at the time the application sends a response back to the MCS, then the MCS routes it back to the same station. But, if the application requires that the response be sent to some other station, then that is accomplished by merely changing the LSN field to the correct value for the other station.

TRANSACTION-BASED ROUTING.

By using the FAMILY statement of NDL, the network designer can associate a list of stations with a remote file name. When an application program successfully opens a remote file, the stations in the family associated with that file become, in a sense, "attached" to that program. From that time on, the Network Controller routes messages from that family of stations only to the program that opened the file.

However, GEMCDS offers an optional alternative to station attachment: Transaction-Based Routing (TBR). Messages may contain transaction codes anywhere in the text. TBR allows a group of transaction codes to be associated with each application program, so that a message containing a specific transaction code is always routed to the same program, regardless of the message source. A station can then send a message to any program (subject to security restrictions) by including in the message one of the transaction codes associated with that program. If a message does not have a transaction code or the transaction code is invalid, the message is sent to the program which is attached to the originating station. If no program is attached to that station, the message is rejected.

ACCESS CONTROL.

In GEMCDS there are two types of access control: access security and process security. Access security is designed to prevent unauthorized persons from using the system. Process security limits the functions that authorized persons are allowed to perform. A specific MCS may be generated to contain logic for access security alone or for both access security and process security.

ACCESS SECURITY.

Access security is implemented through sign-on and sign-off messages. Some stations may already be physically secure; therefore, signing on is not necessarily required for all stations. Each installation may specify which stations do not require signing on.

The sign-on message requires that each potential station user supply a user-identification code (access key). A list of active access keys is defined in TCL and is stored in the MCSTIC file. Moreover, for each station that must be signed on, TCL can be used to specify which access

**STATION Section
cont**

keys are to be recognized. The TCL compiler can be used to update this information. At any specific time a user station may be either enabled or disabled. If disabled, then that user is not allowed to sign on until enabled again.

A user is allowed to sign on only if all of the following conditions hold:

- a. The station requires signing on.
- b. The station is not already signed on.
- c. The user enters a valid access code.
- d. The access code entered is to be recognized at the station being signed on.

A user is allowed to enter a data message only if one of the following conditions holds:

- a. The terminal does not require signing on.
- b. The user is signed on.

The same access key can be used to sign on at several stations simultaneously.

PROCESS SECURITY.

GEMCOS offers two types of process security:

- a. Transaction security - for limiting which transaction codes can be entered by a signed-on user.
- b. Program security - for limiting which programs can be used by a signed-on user. Program security is used when messages do not have transaction codes.

As access codes are defined in TCL, they are associated with a list of valid transaction codes and/or program identifiers. Once signed on, a particular access key is restricted to those transactions and/or programs with which it was associated. Hence, each access code can be limited to any station, program, or trancode combination.

NETWORK ADMINISTRATION.

Although the MCS and Network Controller automatically control many aspects of the network, some conditions still require human intervention. For this reason, GEMCOS supports the CONTROLSTATION concept. The supervisory console can always be used as a Control station. In addition, one remote station can be designated as the Control station.

A control station administers the network through Network Control Commands (NCCs). GEMCOS recognizes Network Control Command syntax for the following functions:

- a. Access control.

- 1) Enable and disable users.
 - 2) Sign on and off.
- b. MCS control.
- c. Message control.
- 1) Reroute messages.
 - 2) Retrieve queued messages.
 - 3) Send messages to other stations.
- d. Program control.
- 1) Execute and terminate Application Programs.
 - 2) Report program status.
- e. Station status. Report and change status of stations.

The MCS need not contain logic to execute all these commands since there are TCL parameters to specify which commands are needed.

Use of Network Control commands is restricted by the GEMCOS security subsystem. Some commands can be entered by any one at any station (i.e., the sign-on and sign-off commands), but many commands can only be entered at the Console station. All Network Control Commands can always be entered at the console keyboard without restriction.

ERROR HANDLING.

The GEMCOS error handling subsystem provides the necessary logic to handle error conditions which are not directly related to the applications task at hand (thereby relieving the application programmer of that burden). With GEMCOS, enough automatic action is taken to keep the data communications system running, and the error condition is communicated to the person who has the power to correct the problem.

GEMCOS distinguishes three error categories:

- a. Errors made by a station operator.
- b. Persistent data communications errors.
- c. System errors.

When a GEMCOS MCS detects errors made by a station operator, error messages are sent back to that operator. For other kinds of errors, messages are sent to the Control station or to the console printer, if the Control station is not available. Network Control Commands may then be used to help diagnose the problem or circumvent it.

The Network Controller handles transient data communications errors, but persistent errors are reported to the MCS. The MCS not only reports such errors to the Control station but also keeps error statistics. Statistics are accumulated by station. They can be retrieved by using Network Control commands.

System errors result either from input/output errors on peripheral devices used by the MCS or from software problems in the MCS, the Network Controller, or the applications programs. When a system error is detected, the MCS reports the error to the Control station. For serious errors, the MCS also produces a dump of its tables for debugging purposes. The MCS is designed, however, to continue running unless the Control station or system operator discontinues it.

NETWORK RESTORATION.

The purpose of network restoration is to bring the Network Controller up-to-date with MCS data on network status. Network status consists of information such as the current physical address of a station, or whether a station is "alive" or "dead." The MCS stores current network status information on disk file MCSTIC (at a location that is protected from abnormal program termination). Network restoration is done automatically.

Each time the MCS is executed, it uses the status data from the MCSTIC file to generate commands for the Network Controller. The Network Controller uses these commands to update its tables in main memory.

MESSAGE RECOVERY.

GEMCOS provides message recovery at either the program or data base level via an audit trail to help application programs recover from failure. Through TCL parameters, an MCS can be created with just the audit feature or with both audit and recovery.

AUDIT.

When the audit option is used, the MCS keeps an audit trail of all messages sent to an Application Program. Output messages may be audited for nonsynchronized recovery, but must be audited for synchronized recovery. Messages are identified by a sequence number which the MCS assigns. The Common-area header is used to communicate the sequence number to Application Programs. For synchronized recovery (total version only), a data base sequence number is also assigned upon receipt of the output message.

The audit trail is written on a disk file. When the audit file becomes filled, it is closed and a new one is opened. The file is then available for copying to another device. Each new audit file has a file identifier which is different from the last.

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 simple queue restoration technique to an automatic data base rollback and synchronization scheme. Once again, the emphasis is on providing users with the flexibility of easily adapting to meet a broad range of on-line data processing requirements.

CONTROLLED SHUTDOWN.

When a data communications system terminates operations, messages may be lost in the process of terminating unless special care is taken. In GEMCOS, system shutdown consists of the following steps:

- a. A message is sent to all stations, informing them that shutdown has begun.
- b. Further input is disabled.
- c. The MCS causes an End-Of-File condition on all remote files.
- d. Any messages that may remain in the queues of the Network Controller are recalled and printed on a line printer.
- e. The MCS terminates.

Any messages that could not be delivered to their destinations are accounted for on the printer listing.

MERGEABLE_EXTERNAL_SOURCE_STATEMENTS.

Mergeable External Source Statements (MESS) are used for specialized requirements which demand deviation from the standard GENCOS logic. User-written MESS routines can be merged into key locations in the MCS logic. Each routine must be a procedure in User Programming Language (UPL). All MESS routines are optional. Most MESS routines can either replace or supplement standard GENCOS logic.

The MESS procedures can be inserted in the following code locations (the intended function of each procedure is explained, but there is practically no limit to the functions that can be coded):

- a. Receiving message from station - for formatting, paging, or routing.
- b. Receiving message from program - for formatting, paging, or routing.
- c. Processing Network Control commands - for extending or replacing the capabilities for network control.
- d. Auditing - for replacing or supplementing the standard audit feature.
- e. Error handling - for extending the standard error-handling logic.
- f. Opening - for processing required at system startup.
- g. Closing - for processing required at system shutdown, such as closing files used by other MESS procedures.
- h. Recalling messages - for disposing of unsent messages when the system is shut down.
- i. Initiating recovery - for processing the request of an Application Program for recovery.
- j. Recovery - for replacing the standard recovery logic.
- k. Remote File Open - for processing that is required when an Application Program opens a remote file.
- l. Remote File Close - for processing that is required when an Application Program closes a remote file.

The source code for MESS procedures is submitted as part of the user's TCL source file. The TCL compiler merges these procedures into the correct places in the MCS logic.

DEBUGGING AIDS.

GEMCOS offers two kinds of debugging aids: a data dump and a logic flow monitor. The existence of these features is controlled by TCL parameters.

DATA DUMP.

The data dump provides a snapshot of the state of the MCS and its environment. It displays the contents of the tables, certain significant data items, and the message work area. It can be used for debugging purposes and for reporting statistical information maintained in the tables. A data dump can be created on demand, via a Network Control command, or automatically, when the MCS detects a serious error.

MONITOR TRACE.

The monitor is a procedure within the MCS which produces a listing that can be used to trace logic flow. Calls on the monitor are made at the entrance to nearly every procedure and at other key points in the MCS. Since MESS procedures can also call on the monitor, the monitor is a valuable tool for debugging interfaces between MESS code and the standard MCS.

The monitor listing displays:

- a. An identification of the procedure now executing.
- b. An identification of the procedure which called the current procedure.
- c. Any information that is pertinent to the current procedure.
- d. The sequence number in the MCS source code file at the point where the monitor was called.

Since the monitor is a generative option, the MCS used for normal operation does not have to carry the overhead of monitor logic. Users can generate a second MCS which is exactly like the production MCS except that the MONITOR generation parameter is set. This second MCS would be used only if problems arise which cannot otherwise be diagnosed.

When an MCS is generated with the MONITOR parameter set, the monitor listing can be turned on and off either dynamically, by a Network Control command, or between executions of the MCS by the TCL compiler. Furthermore, the monitor can be selected for individual procedures, or for only those procedures suspected of being related to a problem.

SCREEN_WRAPAROUND.

To provide Application Programs with enhanced terminal independence, GEMCOS can automatically segment messages to fit the buffer of the destination station. Instead of transmitting a message that cannot be accepted by a station because of its length, the MCS can break the message into two or more transmissions.

The message is segmented on word boundaries whenever possible. The segment will be as long as is possible without splitting a word or exceeding the buffer at the station. If the message contains a string of nonblank elements that is greater than the station buffer size, it is necessary to break the string into one or more transmissions. Using screen wraparound is not recommended with messages that contain formats. A formatted message could be segmented in the middle of a forms field. A warning message is sent to the Control station when screen wraparound sends a formatted message to a station.

Code for screen wraparound is generated when the station screensize is declared larger than the Global MAXTEXTSIZE.

SUPERVISORY_MCS.

GEMCOS provides a supervisory MCS function allowing stations to be switched between MCS programs. The user may, for example, use CANDE and ODESY, which are both MCS programs and switch between them as the need arises. Without a supervisory MCS, both CANDE and ODESY would have to be shut down and restarted to switch stations, resulting in considerable inconvenience to the rest of the station operators. However, with GEMCOS Network Control commands, station operators can switch from one subordinate MCS to another without interrupting the other operators.

REMOTE_PROGRAM_EXECUTION.

Among the on-line applications of a given installation, there may be programs which would be most conveniently used if they could be executed remotely. GEMCOS provides this capability. All on-line programs need not be executed remotely, however. There are situations where it is desirable to have centralized control over program execution. GEMCOS also provides the ability to exercise centralized control via the supervisory console or a remote control station.

FORMATTING.

One of the major problems of on-line programming is the human interface. Since the system can interact with many individuals through stations, some of whom may only spend a small percentage of time working with a terminal, it is important that the input and output be as "humanly readable" as possible. Proper formatting of information exchanged between the terminal and the system provides this readability.

When approaching the problem of formatting, the strategy employed must be chosen carefully. Application Programs can be written to expect and create readable messages. However, they then become much more complex, especially if several terminal devices are involved. Moreover, if existing terminal devices are replaced with others, the application programs must be converted. Application Programs are much easier to write and maintain if they read and write records instead of messages that can be clearly understood by people.

The GEMCOS formatting function of the advance version was designed with these aspects in mind. It can be used to support forms requests, enhance the readability of message text and ensure Application Program device independence.

SECTION 3

TRANSACTION CONTROL LANGUAGE

GENERAL DISCUSSION OF TRANSACTION CONTROL LANGUAGE (TCL).

The B 1800/B 1700 Transaction Control Language is classified as a descriptive language. It is a high-level language providing a simple means of selecting required MCS functions and describing on-line system relationships. The result of a TCL compilation is an MCS program composed of GEMCOS intrinsics and a data file consisting of on-line relationships. (An MCS is a program which works closely with a Network Controller to provide functions such as remote file control, error handling, access control, audit, routing, formatting, etc., for on-line application programs.) The size limitations of the TCL compiler are given in appendix G.

If the requirements of the MCS or the relationships with which it operates change, a new system may be easily obtained by recompiling.

The TCL compiler is found on a GEMCOS release tape in a file labeled MCSTCL. When MCSTCL is executed, it awaits a TCL source deck labeled MCSIN. The cards which compose a TCL source deck are similar to those of a UPL source deck:

- a. Columns 73 through 80 are reserved for sequence numbers.
- > b. Comments may occur on any card following a "Z".
- c. Statements may begin in any column.

The TCL compiler does not permit a continuation from one card to the next. If a string is begun on a card, it must end on that card.

SYNTAX CONVENTIONS.

The following is a discussion of the Backus-Naur form 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:

- a. 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
- b. The symbol (::=) means "is defined as". The metalinguistic variable to the left of this symbol is defined by the metalinguistic formula on its right.

- c. The slash (/) means "or". It separates alternate definitions of a metalinguistic variable. The conventional symbol of a vertical bar is not used in this document to represent the word "or".
- d. Brackets ([]) are used to enclose metalinguistic variables which are defined by the meaning of the English language expression contained within the brackets. This formulation is used only when it is impossible or impractical to use a metalinguistic formula.

METALINGUISTIC FORMULAE.

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.

Any mark or symbol in a metalinguistic formula which is not one of the metalinguistic symbols is equivalent to itself. The juxtaposition of the metalinguistic variables and/or symbols in a metalinguistic formula denotes the juxtaposition of those elements in the construct indicated.

An example of a metalinguistic formula is:

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

This metalinguistic formula is read:

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 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) Set, Description, Section, Statement, Command or Declaration, 2) Syntax, 3) Semantics, 4) Pragmatics, if applicable, and 5) Example(s), 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:

<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>	::= /
<single space>	::= [one horizontal position]
<space>	::= <single space> / <space> <single space>
<character>	::= <string character> / <string bracket character>
<string character>	::= <letter> / <digit> / <special character>
<string bracket character>	::= "
<empty>	::= [the null string of symbols]

Semantics:

The character set for the handler 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 components are the prime structures of the
language.**

**Basic Components
cont**

IDENTIFIERS.

Syntax:

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

Semantics:

**The maximum length of an identifier is 30 characters. Spaces may
not appear as part of an identifier.**

**Basic Components
cont**

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>	::= <string> / <hexadecimal string>
<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 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>"
<EBCDIC CODE>	::= 8 / <empty>
<hexadecimal code>	::= 4
<one-byte string>	::= <EBCDIC unit string> / <hex unit string>

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.

**Basic Components
cont**

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 statement> /
                        <CONTROL statement>
                        <MCSTIC FILE NAME statement>
                        <FORMAT FILE NAME statement>
                        <GLOBAL section>
                        <DEFINITION section>

```

Semantics:

In order to create an MCS with B 1800/B 1700 GEMCDS, the user must execute MCSTCL. First the user must load MCSGTS, MCSGO and MCSTCL from the B 1800/B 1700 GEMCDS release tape. MCSTCL can then be executed by reading in a card deck constructed as follows:

```

?EX MCSTCL
?DATA MCSIN
.
.
.
<Deck description>
.
.
.
?END

```

As soon as the compilation begins, MCSTCL reads MCSIN, and writes MCSLST on a line printer. MCSLST consists of a listing of the <Deck description> along with any syntax errors. If there are no syntax errors, MCSTCL takes the actions specified in the <CONTROL statement>.

The user could create and maintain a TCL source file with CANDE instead of a card deck. The CANDE default file type should be used. To run MCSTCL with TCL source file created through CANDE, enter the following:

```
EX MCSTCL FILE MCSIN NAME <user's CANDE file name> DSK DEF;
```

DECK DESCRIPTION
cont

Example:

```
?EX MCSTCL
?DATA MCSIN
CONTROL = GENERATE, LIST, COMPILE.
GLOBAL:
  MONITORTRACE = TRUE.
  NCCOKRESPONSE = "$OK$".
  CONTROLSTATIONS = TD800A.
BEGIN
PROGRAM PAYROLL USER:
  TITLE = PAYROLL.
  TRANCODE = UPDATE(1,1).
  TRANCODE = INQ(1,2).
PROGRAM INVENTORY USER:
  TITLE = INVNT.
  TRANCODE = RCV (2,1), SHIP (2,2).
PROGRAM GAME UTILITY:
  TITLE = MAZE/GAME.
  INTERFACE = NONPARTICIPATION.
STATION TD800A:
STATION TD800B:
STATION TD800C:
STATION TD700A:
STATIC DECLARATIONS:
  DECLARE 01 MESS.STRUCTURE      CHARACTER(5),
          02 MESS.ITEM.1        CHARACTER(3),
          02 MESS.ITEM.2        CHARACTER(2);
ENDSOURCECODE.
PROCEDURE SETVALUES:
  PROCEDURE MESS.SET.VALUES;X
    MESS.ITEM.1 := "AAA";X
    MESS.ITEM.2 := "BB";X
  END MESS.SET.VALUES;X
ENDSOURCECODE.
END.
?END.
```

CONTROL STATEMENT.**Syntax:**

```

<CONTROL statement> ::= CONTROL = <control list>.

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

<control task>      ::= LIST / GENERATE / REGENERATE / COMPILE /
                        UPDATEFMT

```

Semantics:

The <CONTROL statement> defines the task(s) to be performed during a run of MCSTCL. The <control list> defines the individual task or combination of tasks.

LIST causes a hard-copy record of the user's data communication system description to be written to a line printer. The listing is labeled MCSRPT. If LIST is the only option in the <control list>, the <GLOBAL section> and the <DEFINITION section> are not required; however, the MCSTIC file must be available to MCSTCL.

GENERATE causes MCSTCL to create a disk file labeled MCSTMP and ZIP MCSGO. MCSGO uses MCSTMP and MCSGTS to create MCSSRC, the user's MCS source-code file. In addition, when GENERATE appears in the <CONTROL statement>, a disk file labeled MCSTIC is written. MCSTIC contains customized tables consisting of the user's data communication system network relationships. The MCSTIC file must be present when executing a B 1700 GEMCDS-generated MCS. When GENERATE is in the <control list>, both the <GLOBAL section> and <DEFINITION section> must be present.

REGENERATE causes MCSTCL to create a new MCSTIC file from an old one. This option should be used if a station, transaction, program, or access key is to be added or changed. REGENERATE neither writes MCSCRD nor ZIP-executes MCSGO, thus saving machine time. When REGENERATE is in the <control list>, both the <GLOBAL section> and the <DEFINITION section> must be present. If MCSTCL determines (while modifying an existing MCSTIC file) that the MCS code file is no longer compatible, it produces a syntax error and the regeneration does not occur. This happens when, for example, AUDIT was not specified in the original GENERATE run, but appears in the REGENERATE run.

**CONTROL Statement
cont**

NOTE

During a REGENERATE run, the station network control information, which is used to bring stations back to their last running state, is not copied from the old MCSTIC file to the new one. Therefore, after a regeneration, stations in the network have those attributes specified in NDL which do not reflect the accumulated changes caused by GEMCOS Network Control Commands. Moreover, the audit file number is reset to zero; all existing audit files are no longer valid.

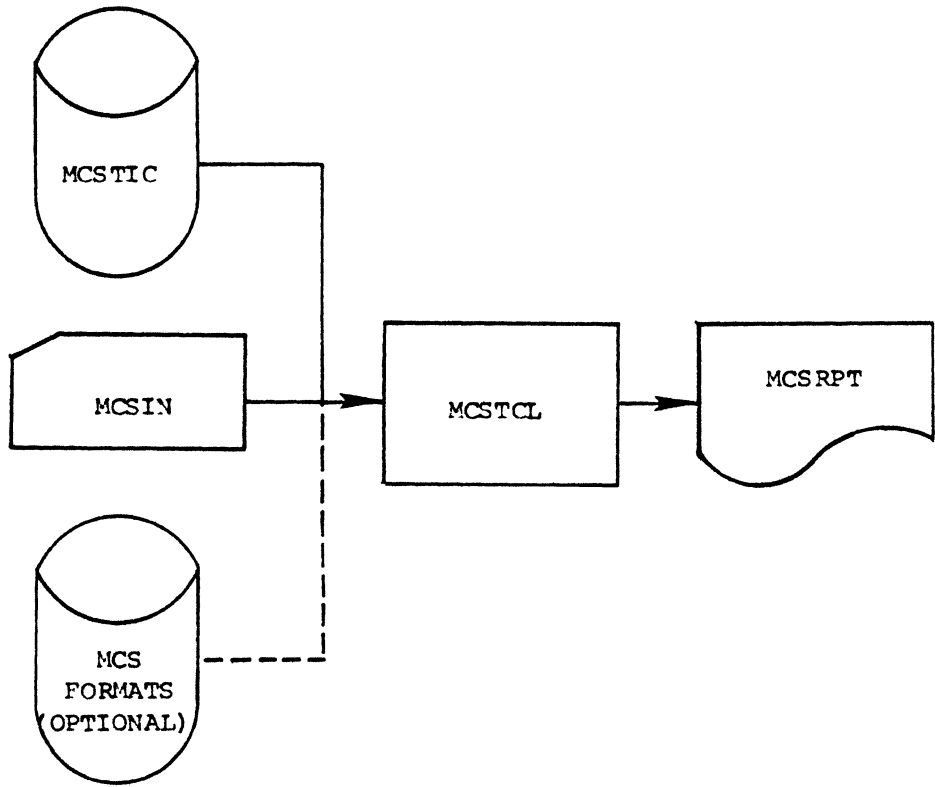
COMPILE causes MCSTCL to instruct MCSGO to ZIP-execute the UPL compiler to create MCSSRC/object from MCSSRC. If COMPILE appears in the <control list>, GENERATE must also appear.

UPDATEFMT facilitates recompilation of the TCL FORMAT section without requiring generation or regeneration. The format section can be recompiled while the MCS is operating and without affecting the MCSTIC file. Only previously compiled functions and formats can be modified. The recompiled functions and formats are copied into the format file, MCSFORMATS. Programs and stations have access to the new copy of the format through the *UPD network control command, entered from the control station or the SPD.

Examples:

CONTROL = LIST.
CONTROL = REGENERATE, LIST.
CONTROL = GENERATE, LIST, COMPILE.
CONTROL = UPDATEFMT.

Figures 3-1 through 3-4 illustrate which files are created and accessed by the TCL compiler (MCSTCL) when the previously listed example control statements are present.



**Figure 3-1. Files Created by TCL
Compiler When CONTROL = LIST**

CONTROL Statement
cont

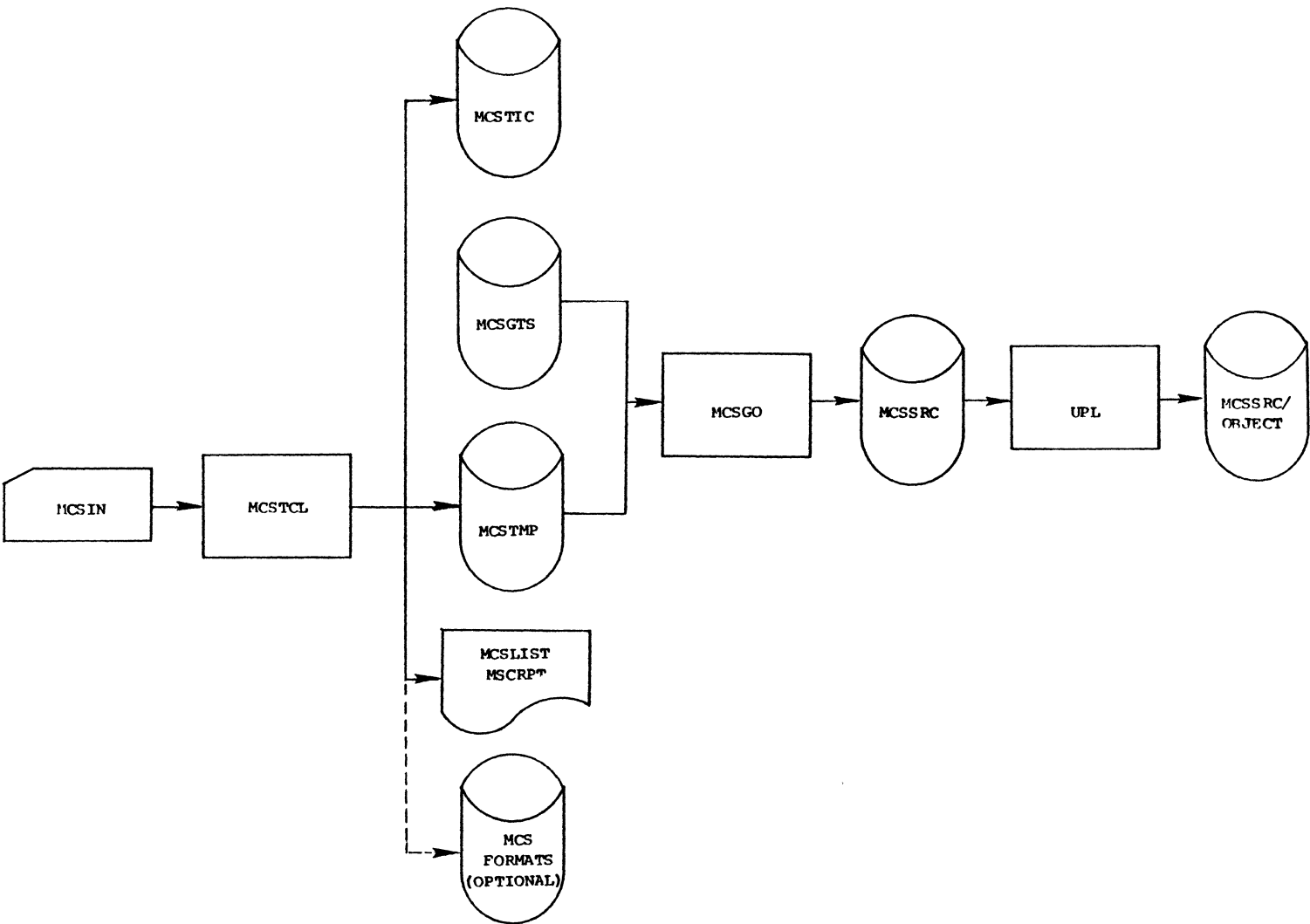
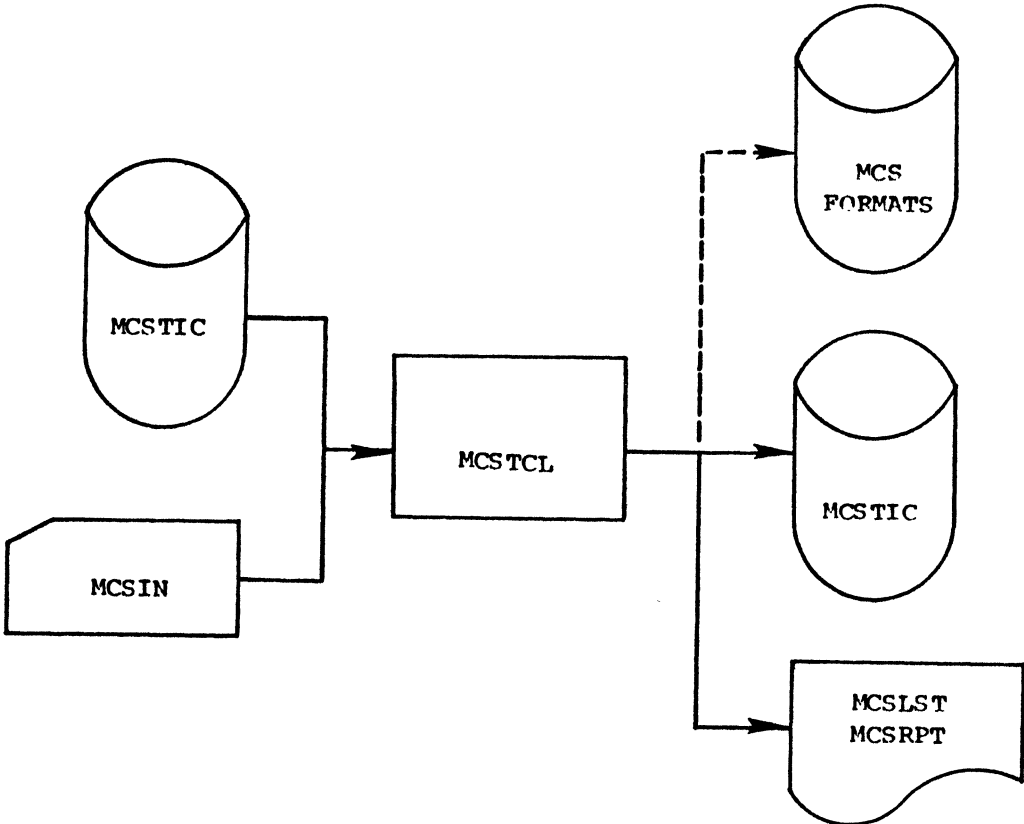


Figure 3-2. Files Created by TCL
Compiler When CONTROL =
GENERATE, LIST, or COMPILE



**Figure 3-3. Files Created by TCL
Compiler When CONTROL =
REGENERATE or LIST**

MCSTIC FILE NAME Statement

MCSTIC FILE NAME STATEMENT.

Syntax:

**<MCSTIC FILE NAME statement> ::= MCSTICFILENAME = <file-ID>./
<empty>**

Semantics:

The <MCSTIC FILE NAME statement> allows for the specification of the MCSTIC file name. The <MCSTIC FILE NAME statement>, if present, must appear after the <CONTROL statement> and before the <GLOBAL section>. <File-ID> is a B 1800/ B 1700 file identifier. By default, MCSTICFILENAME is MCSTIC.

Examples:

**MCSTICFILENAME = MYMCSTIC.
MCSTICFILENAME = TEST/MYMCSTIC.
MCSTICFILENAME = PACKB/TEST/MYMCSTIC.**

FORMAT FILE NAME STATEMENT.

Syntax:

```

<FORMAT FILE NAME statement> ::= FORMATFILENAME = <file-ID>. /
                                     <empty>

```

Semantics:

The <FORMAT FILE NAME statement> is used to change the name of the MCSFORMATS file. This statement only functions in the advanced and total versions of GENCOS. It must immediately follow the <MCSTIC FILE NAME statement> and precede the <GLOBAL section>. <File-ID> is a B 1800/B 1700 file identifier. By default, FORMATFILENAME is MCSFORMATS.

Examples:

```

FORMATFILENAME = ALLFORMATS.
FORMATFILENAME = TEST/FORMATS.
FORMATFILENAME = GEMPAC/LIVE/FORMATS.

```

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> ::= <CODE GENERATION statement>
                       <MISCELLANEOUS PARAMETER statement>

<CODE GENERATION statement> ::= <CHANGE REQUESTS statement> /
                                <DATA DUMP statement> /
                                <MESSAGE BROADCAST statement> /
                                <MESSAGE RECALL statement> /
                                <PROGRAM BOJ EOJ statement> /
                                <MONITOR TRACE statement> /
                                <STATUS REPORTS statement> /
                                <SYSTEM HALT statement> /
                                <COMPILE OPTIONS statement> /
                                <OBJECT CODE FILE NAME statement> /
                                <SOURCE CODE FILE NAME statement> /
                                <NAME STACK ENTRIES statement> /
                                <VALUE STACK BITS statement> /
                                <CONVERSATIONLIMIT statement>

<MISCELLANEOUS PARAMETER statement>
    ::= <NCC OK RESPONSE statement> /
        <SIGNAL CHARACTER statement> /
        <AUDIT RECORD SIZE statement> /
        <AUDIT PAGE SIZE statement> /
        <AUDIT FILE PACK ID statement> /
        <CHECKPOINT INTERVAL statement> /
        <MAX TEXT SIZE statement> /
        <QUEUE DEPTH statement> /
        <QUEUE BUFFERS statement> /
        <QUEUE NAME statement> /
        <SIMULATION statement> /
        <MONITOR TRACE ON statement> /
        <FORMAT AND FUNCTION statement list> /
        <RECALL PROGRAM statement> /
        <CONTROLSTATIONS statement>

```

Semantics:

The <GLOBAL section> is composed of two types of <GLOBAL statements>: <CODE GENERATION statements> and <MISCELLANEOUS PARAMETER statements>. Any given <GLOBAL statement> may only occur once in the <GLOBAL section> with the exception of the format and function statements.

There are two types of <CODE GENERATION statements>. First, there are <CODE GENERATION statements> which cause optional MCS intrinsics to be generated into the MCS source file; they can take on a true or false value. Optional MCS intrinsics include code to support change commands, the data dump command, message broadcast, message recall, program control commands, the monitor trace, status commands, system shutdown, audit, output audit and queue restoration. Second, there are <CODE GENERATION statements> which control the names of GEMCOS files, UPL compiler options, and object code memory size requirements. It is important to note that both types of <CODE GENERATION statements> directly affect the MCS source and/or object code files. Therefore, if a <CODE GENERATION statement> is modified, GENERATE and COMPILE should appear in the <CONTROL statement> since new source and object code files are required; otherwise, MCSTCL detects an object code file, MCSTIC file incompatibility error.

<MISCELLANEOUS PARAMETER statements> specify various attributes of a running GEMCOS MCS such as the signal character, Control station, Network Control Command response, etc. Except for the <MAX TEXT SIZE statement>, the <AUDIT PAGE SIZE statement> and the <CONTROL STATIONS statement>, <MISCELLANEOUS PARAMETER statements> may be safely changed in a REGENERATE MCSTCL run.

Example:

```

GLOBAL:
PROGRAMBOJEDJ = TRUE.
MONITORTRACE = FALSE.
COMPILEOPTIONS = "LIST SINGLE".
QUEUEBUFFERS = 3.
DATADUMP = TRUE.

```

CHANGE REQUESTS STATEMENT.

Syntax:

```
<CHANGE REQUESTS statement> ::=  
    CHANGEREQUESTS = <logical value>. /  
    <empty>
```

Semantics:

The <CHANGE REQUESTS statement> determines whether the GEMCDS MCS is to contain the logic to support the following seven Network Control Command change requests:

- a. CHANGE STATION ADDRESS (CSA).
- b. CHANGE STATION DIAGNOSTIC (CSD).
- c. CHANGE STATION FREQUENCY (CSF).
- d. CHANGE STATION MAXIMUM RETRY (CSM).
- e. CHANGE STATION QUEUE (CSQ).
- f. CHANGE STATION READY (CSR).
- g. CHANGE STATION TRANSMISSION NUMBER (CST).

When MONITORTRACE equals TRUE, the CHANGE MONITOR FLAG (CMF) command becomes the eighth change request and CHANGEREQUEST defaults to TRUE. Otherwise, CHANGEREQUESTS defaults to FALSE.

Example:

```
CHANGEREQUESTS = TRUE.
```

DATA DUMP STATEMENT.

Syntax:

<DATA DUMP statement> ::= DATADUMP = <logical value>. / <empty>

Semantics:

The <DATA DUMP statement> indicates whether the code to create a dump of internal MCS variables is present. If DATADUMP equals TRUE, the REPORT DATA DUMP (RDM) command is recognized. By default, DATADUMP equals FALSE.

Example:

DATADUMP = FALSE.

MESSAGE BROADCAST STATEMENT.

Syntax:

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

Semantics:

The <MESSAGE BROADCAST statement> specifies if the code to support the BROADCAST (BRC) Network Control Command is to be generated. By default, MESSAGEBROADCAST equals FALSE.

Example:

```
MESSAGEBROADCAST = TRUE.
```

MESSAGE RECALL STATEMENT.

Syntax:

```
<MESSAGE RECALL statement> ::=  
    MESSAGERECALL = <logical value>. /  
    <empty>
```

Semantics:

The <MESSAGE RECALL statement> indicates whether the code to support the POP QUEUE (PQ) Network Control Command will be generated. MESSAGERECALL equals FALSE by default.

Example:

```
MESSAGERECALL = TRUE.
```


**GLOBAL Section
cont**

PROGRAM BOJ EOJ STATEMENT.

Syntax:

```
<PROGRAM BOJ EOJ statement> ::=  
PROGRAMBOJEOJ = <logical value>. /  
<empty>
```

Semantics:

The <PROGRAM BOJ EOJ statement> determines if the EXECUTE PROGRAM (EX), HALT APPLICATION PROGRAM (HAP), and CLEAR BUSY FLAG (CBF) Network Control Commands are to be supported. By default, PROGRAMBOJEOJ equals FALSE. This statement should be set to TRUE if Utility Programs are to be generated into the MCS.

Example:

```
PROGRAMBOJEOJ = FALSE.
```

MONITOR TRACE STATEMENT.

Syntax:

```
<MONITOR TRACE statement> ::=  
    MONITORTRACE = <logical value>. /  
    <empty>
```

Semantics:

The <MONITOR TRACE statement> specifies whether to generate logic for the Debug Monitor. When MONITORTRACE is set, CHANGEREQUESTS becomes TRUE by default to include the CMF Network Control Command. However, if CHANGEREQUESTS equals TRUE and MONITORTRACE equals FALSE, the CMF Network Control Command is not recognized. By default, MONITORTRACE equals FALSE.

Example:

```
MONITORTRACE = TRUE.
```

STATUS REPORTS STATEMENT.

Syntax:

```
<STATUS REPORTS statement> ::=  
    STATUSREPORTS = <logical value>. /  
    <empty>
```

Semantics:

The <STATUS REPORTS statement> determines whether to include the logic to support the following five Network Control Command status report requests:

- a. REPORT FILE STATUS (RFS).
- b. REPORT PROGRAM COUNTERS (RPC).
- c. REPORT PROGRAM STATUS (RPS).
- d. REPORT STATION COUNTERS (RSC).
- e. REPORT STATION STATUS (RSS).

STATUSREPORTS equals FALSE by default.

Example:

```
STATUSREPORTS = FALSE.
```

SYSTEM HALT STATEMENT.

Syntax:

```
<SYSTEM HALT statement> ::= SYSTEMHALT = <logical value>. /  
                             <empty>
```

Semantics:

The <SYSTEM HALT statement> specifies if the code for handling the HALT (HLT) Network Control Command is to be generated. If SYSTEMHALT is set TRUE, CHANGEREQUESTS is set TRUE. SYSTEMHALT equals FALSE by default.

Example:

```
SYSTEMHALT = TRUE.
```

COMPILE OPTIONS STATEMENT.

Syntax:

```
<COMPILE OPTIONS statement> ::=  
    COMPILEOPTIONS = <string>. / <empty>
```

Semantics:

The <COMPILE OPTIONS statement> allows for the specification of UPL compiler control statements when COMPILE appears in the <CONTROL statement> (refer to the Burroughs B 1700 Systems User Programming Language (UPL) Reference Manual, form 1067170, for a complete description of available options). <String> must begin and end with a quote and must not exceed 65 characters. By default, COMPILEOPTIONS is set to NO LIST NO_DUPLICATES SUPPRESS USEDOTS.

Examples:

```
COMPILEOPTIONS = "LIST SINGLE".  
COMPILEOPTIONS = "LIST XMAP XREF".
```

OBJECT-CODE FILE NAME STATEMENT.

Syntax:

```
<OBJECT CODE FILE NAME statement> ::=  
    OBJECTCODEFILENAME = <file-ID>. /  
    <empty>
```

Semantics:

The <OBJECT-CODE FILE NAME statement> allows for the specification of the MCS object code file name when COMPILE appears in the <CONTROL statement>. <File-ID> is a B 1800/B 1700 file identifier. By default, OBJECTCODEFILENAME is MCSSRC/OBJECT.

Examples:

```
OBJECTCODEFILENAME = MCS.  
OBJECTCODEFILENAME = INVENTORY/MCS.
```

SOURCE-CODE FILE NAME STATEMENT.

Syntax:

```
<SOURCE CODE FILE NAME statement> ::=  
    SOURCECODEFILENAME = <file-ID>. /  
    <empty>
```

Semantics:

The <SOURCE-CODE FILE NAME statement> allows for the specification of the MCS source code file name when GENERATE appears in the <CONTROL statement>. <File-ID> is a B 1800/B 1700 file identifier. By default, SOURCECODEFILENAME is MCSSRC.

Examples:

```
SOURCECODEFILENAME = MCS/SOURCE.  
SOURCECODEFILENAME = SOURCE/FILE.
```

NAME-STACK ENTRIES STATEMENT.**Syntax:**

```
<NAME STACK ENTRIES statement> ::=  
    NAMESTACKENTRIES = <integer>. /  
    <empty>
```

Semantics:

The <NAME STACK ENTRIES statement> specifies the maximum number of name-stack entries that need to be reserved for variables declared by user-written code. This parameter is used to ensure that stack sizes are large enough to execute an MCS which contains user-written code. If the value assigned in this statement is not large enough, a name or value-stack overflow error may occur when the MCS is executed. Name-stack entries are used to store information concerning variables. One name-stack entry is used for each data name that appears in a <DECLARE statement>. If a data name refers to an array, it requires two name-stack entries. By default, NAMESTACKENTRIES is set to 0.

To achieve optimal memory use, GEMCDS estimates the name-stack space required for its variable declarations and overrides the UPL compiler defaults. If user code is being included, NAMESTACKENTRIES should be set appropriately. The value given to NAMESTACKENTRIES is added to the GEMCDS estimate. If user-written code is not included, the <NAME STACK ENTRIES statement> may be ignored.

Examples:

```
NAMESTACKENTRIES = 25.  
NAMESTACKENTRIES = 100.
```


VALUE-STACK BITS STATEMENT.

Syntax:

```
<VALUE STACK BITS statement> ::=  
    VALUESTACKBITS = <integer>. /  
    <empty>
```

Semantics:

The <VALUE STACK BITS statement> specifies the maximum number of value-stack bits that are needed as a result of user-code data-name declarations. This parameter is used to ensure that stack sizes are large enough to execute an MCS which contains user-written code. If the value assigned in this statement is not large enough, a name or value-stack overflow error may occur when the MCS is executed. The value of a variable which requires 24 or less bits requires no room on the value stack. However, if a variable requires more than 24 bits, or if the variable refers to an array, space must be reserved on the value stack for that variable. By default, VALUESTACKBITS equals zero.

In a fashion similar to the <NAME STACK ENTRIES statement>, the <VALUE STACK BITS statement> enables GEMCOS to achieve optimized memory use. GEMCOS estimates the value-stack space required for its variables and overrides the UPL compiler defaults. If user code is included, VALUESTACKBITS should be set appropriately. The number assigned to VALUESTACKBITS is added to the GEMCOS estimates. If user-written code is not included, the <VALUE STACK BITS statement> may be ignored.

Examples:

```
VALUESTACKBITS = 1000.  
VALUESTACKBITS = 256.
```

CONVERSATIONLIMIT STATEMENT.**Syntax:**

```
<CONVERSATIONLIMIT statement> ::= CONVERSATIONLIMIT = <integer> /  
                                <empty>
```

Semantics:

The <CONVERSATIONLIMIT statement> allows the user to specify the maximum number of stations that may converse concurrently. The integer specified must not exceed the number of stations declared in TCL. The maximum limit allowed by GENCDS is 64. If there are no CONVERSATIONSIZE statements declared for programs in the TCL, the default value is zero. That is, no conversation capability exists in the MCS. If conversational programs are present, the default value is the number of stations declared in the TCL. If zero is declared, no conversation capability exists in the MCS.

This statement establishes the number of reserved conversation areas. The number of areas are reserved by powers of 2. When the limit is declared, the nearest 2 to the nth power that is greater than or equal to the limit is the number of areas reserved. Even if the reserved area is larger than the limit, the maximum number of concurrent conversations may not exceed the specified limit. If the limit needs to be increased and the new limit exceeds the number of reserved areas, a GENERATE and re-COMPILE is required.

Examples:

```
CONVERSATIONLIMIT = 8.  
CONVERSATIONLIMIT = 5.
```

NCC OK RESPONSE STATEMENT.

Syntax:

**<NCC OK RESPONSE statement> ::= NCCOKRESPONSE = <string>. /
<empty>**

Semantics:

The <NCC OK RESPONSE statement> defines the message to be returned to a station upon successful completion of a Network Control Command. The <string> must begin and end with a quote and cannot exceed eight characters in length. By default, NCCRESPONSE is \$ (dollar sign).

Examples:

**NCCOKRESPONSE = "NCC OK".
NCCOKRESPONSE = "DONE".
NCCOKRESPONSE = "*OK*".**

SIGNAL CHARACTER STATEMENT.

Syntax:

```
<SIGNAL CHARACTER statement> ::=  
    SIGNALCHARACTER = <character>. /  
    <empty>
```

Semantics:

The <SIGNAL CHARACTER statement> defines the character which, when encountered in the first position of a message, signals the Network Controller and the MCS that the message is a Network Control Command. The character must be a single character enclosed in quotes. By default, SIGNALCHARACTER is "*".

Example:

```
SIGNALCHARACTER = "*".
```

AUDIT RECORD SIZE STATEMENT.

Syntax:

**<AUDIT RECORD SIZE statement> ::= AUDITRECORDSIZE = <integer>. /
<empty>**

Semantics:

The <AUDIT RECORD SIZE statement> controls the size of the audit record by specifying the number of bytes in each record. Increments of 180 are the only allowable values. If a value other than an increment of 180 is specified, a warning is issued and the next highest increment of 180 is selected. By default, AUDITRECORDSIZE equals 180.

Examples:

**AUDITRECORDSIZE = 180.
AUDITRECORDSIZE = 540.**

AUDIT PAGE SIZE STATEMENT.

Syntax:

**<AUDIT PAGE SIZE statement> ::= AUDITPAGESIZE = <integer>. /
<empty>**

Semantics:

The <AUDIT PAGE SIZE statement> controls the size of the audit files by specifying the number of records in each page (i.e., area). There are always 40 pages. By default, AUDITPAGESIZE equals 10000.

Examples:

**AUDITPAGESIZE = 500.
AUDITPAGESIZE = 2000.**

AUDIT FILE PACK ID STATEMENT.

Syntax:

**<AUDIT FILE PACK ID statement> ::= AUDITFILEPACKID = <identifier>.
/ <empty>**

Semantics:

The <AUDIT FILE PACK ID statement> allows MCS audit files to reside on other than the system pack. It is recommended that audit files reside on a user pack to increase throughput and decrease the time spent in audit and recovery. Identifier must be 10 characters or less in length. By default, audit files reside on the system pack.

Examples:

**AUDITFILEPACKID = MCSPACK.
AUDITFILEPACKID = AUDITPACK.**

CHECKPOINT INTERVAL STATEMENT.

Syntax:

**<CHECKPOINT INTERVAL statement> ::= CHECKPOINTINTERVAL = <integer>.
/ <empty>**

Semantics:

The <CHECKPOINT INTERVAL statement> determines the length of time between checkpoints taken by the MCS during auditing. Specifying too small a number causes the MCS to do an excessive number of I/Os, thereby reducing throughput. By default, CHECKPOINTINTERVAL equals 60 (seconds).

Examples:

**CHECKPOINTINTERVAL = 30.
CHECKPOINTINTERVAL = 90.**

MAXIMUM TEXT SIZE STATEMENT.

Syntax:

**<MAX TEXT SIZE statement> ::= MAXTEXTSIZE = <integer>. /
<empty>**

Semantics:

The <MAX TEXT SIZE statement> defines the size, in characters, of the longest message that can pass through the MCS. MAXTEXTSIZE has a direct affect upon the memory requirements of a GEMCOS MCS. It is best to keep MAXTEXTSIZE as low as possible. If the MCS has AUDIT specified as TRUE, the user should never attempt to change MAXTEXTSIZE in a REGENERATE MCSTCL run; otherwise, old audit files will have an incompatible record length. Moreover, an increase in MAXTEXTSIZE usually causes a GENERATE and COMPILE to be required so that the MCS can have a larger value stack. If AUDIT is FALSE, MAXTEXTSIZE can be safely lowered on a REGENERATE MCSTCL run. The default value for MAXTEXTSIZE is 125.

When formatting takes place, resultant messages may contain control characters such as tabs, carriage returns, etc. Each control character takes up one or more positions in the formatted message. An allowance for these characters must be reflected by MAXTEXTSIZE.

Examples:

**MAXTEXTSIZE = 1920.
MAXTEXTSIZE = 300.**

QUEUE DEPTH STATEMENT.

Syntax:

<QUEUE DEPTH statement> ::= QUEUEDEPTH = <integer>. /

Semantics:

The <QUEUE DEPTH statement> specifies the number of messages which may be outstanding in the queue for the MCS. <Integer> may range from 1 to 1023. By default, QUEUEDEPTH equals 20.

Examples:

QUEUEDEPTH = 5.
QUEUEDEPTH = 75.

QUEUE BUFFERS STATEMENT.

Syntax:

```
<QUEUE BUFFERS statement> ::= QUEUEBUFFERS = <integer>. /  
                                <empty>
```

Semantics:

The <QUEUE BUFFERS statement> specifies how many memory buffers are available to the MCS queue before messages begin to overflow to disk. The value assigned to QUEUEBUFFERS directly affects the memory requirements of the on-line system. A value too small or too large can degrade system throughput. It is suggested that the user experiment with this statement to find the most efficient value. QUEUEBUFFERS must not have a value greater than QUEUEDEPTH. <Integer> may range from 1 to 16. By default, QUEUEBUFFERS has the value 1.

Examples:

```
QUEUEBUFFERS = 5.  
QUEUEBUFFERS = 8.
```

QUEUE NAME STATEMENT.

Syntax:

**<QUEUE NAME statement> ::= QUEUENAME = <remote file-ID>. /
<empty>**

Semantics:

The <QUEUE NAME statement> specifies the external file name of the MCS queue (i.e., the remote file opened by the MCS). <Remote file-ID> should appear in a FILE statement in the user's Network Definition Language source deck. MCSQUEUE is the default value of QUEUENAME.

Example:

QUEUENAME = MCSRMT.

SIMULATION STATEMENT.

Syntax:

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

Semantics:

The <SIMULATION statement>, when set, causes the MCS to open a queue file instead of the usual remote file. The program MCSSIM can be used instead of the Network Controller to simulate input via the card reader. Output is simulated to a line printer using the MCS Monitor Trace code. The source code for MCSSIM is MCSIMS. SIMULATION equals FALSE by default.

Example:

```
SIMULATION = FALSE.
```

MONITOR TRACE ON STATEMENT.

Syntax:

```
<MONITOR TRACE ON statement> ::=  
    MONITORTRACEON = <logical value>. /  
    <empty>
```

Semantics:

The <MONITOR TRACE ON statement> allows the user to set or reset the debug monitor flags enabling the initialization procedure to be traced. By default, MONITORTRACEON equals FALSE.

NOTE

The CMF command can be used to set or reset any or all of the monitor flags as soon as initialization is complete.

Example:

```
MONITORTRACEON = FALSE.
```

FORMAT AND FUNCTION STATEMENT LIST.

Syntax:

```
<FORMAT AND FUNCTION statement list>
    ::= <function declaration list>
       <format declaration list> /
       <empty>

<function declaration list> ::= <function declaration> /
                               <function declaration list>
                               <function declaration> / <empty>

<format declaration list>  ::= <format declaration> /
                               <format declaration list>
                               <format declaration>
```

Semantics:

In addition to the functional capabilities of the basic version of B 1800/B 1700 GEMCOS, the advanced version includes a Message Formatting module. The Message Formatting module can be used to support forms requests, modify the text of messages, and/or ensure Application Program device independence. Users of the Basic Version will find that an attempt to invoke the formatting capabilities of GEMCOS results in a syntax error.

The Forms Request function provides station operators with the ability to enter a <message-ID> (refer to <DEVICE section>) and to receive in return a formatted screen with blank data fields. Application Programs may also invoke the Forms Request function causing formatted screens with blank data fields to be displayed at stations in the network.

The text of messages entered at stations can be modified, re-arranged and/or supplemented prior to being routed to the appropriate Application Program. This process is referred to as input formatting. The text of messages written by Application Programs can be modified, re-arranged and/or placed into data fields of formatted screens before being sent to stations and are referred to as output formatting.

When a network is comprised of two or more types of terminal devices, the stations may be grouped into several device classifications in the <DEVICE section>. A set of formats is defined for each device classification. When invoked, the formatting module recognizes the device classification of the station involved and applies a format from the set associated with that classification. As a result, messages sent or received by Application Programs can have a standard record layout regardless of the device type of the destination/source station. Moreover, the Application Program need not be affected by the different control characteristics of different devices.

There are two areas of the Transaction Control Language which relate to formatting: the <DEVICE section> and the <FORMAT AND FUNCTION statement list>. The <DEVICE section> is used to identify which messages are to be formatted and with which formats. The <FORMAT AND FUNCTION statement list> is used to define formats and functions. A format specifies how a screen is to be built and/or how the message text is to be modified. A function defines a translate table which can be referred to by a format.

The <FORMAT AND FUNCTION statement list> is composed of a <function declaration list>, which can be <empty>, followed by a <format declaration list>.

FUNCTION DECLARATION.

Syntax:

```

<function declaration> ::= FUNCTION <function part list>.

<function part list>  ::= <function part> /
                          <function part list> , <function part>

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

<function identifier> ::= <identifier>

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

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

<external string>    ::= <string>

<internal string>    ::= <string>

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

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

```

Semantics:

The <function declaration> defines functions which can be used in a translate <item phrase> of a <format declaration>. A <function identifier> is required as the first argument of the translate <item phrase>. The translate <item phrase> allows a format to translate a string of length n into a string of length m where $0 < n < 7$ and $0 < m < 7$. <String> is therefore limited to a maximum of six characters. Up to 1023 functions may be declared.

A <translate pair> associates an <external string> with an <internal string>. On input, an <external string> is translated into the associated <internal string>. On output, an <internal string> is translated into the associated <external string>. When an Application Program deals with the text of a message, it must use an <internal string> in a translate field. When an operator deals with the text of a message at a station, an <external string> is used.

Refer to FORMAT DECLARATION for examples of FUNCTIONS used in FORMAT.

The <justification and fill part> is described in the following example:

Example:

```

FUNCTION GENDER ("MALE": "1", "FEMALE": "2").
FUNCTION DIGITIN ("1": "ONE", "2": "TWO", "3": "THREE"),
DIGITOUT [EXTERNAL: ALPHA, INTERNAL: INTERNAL: INTEGER]
("ONE": "1", "TWO": "2", "THREE": "3").
  
```

As the translate module searches for a match between the source text to be translated and an <internal string>/<external string>, both the source text and the <internal string>/<external string> are placed into character strings of length six for comparison. The <justification and fill part> enables the user to control the placement of the source text and the <internal string>/<external string> into these character strings. If the <justification and fill part> is <empty>, it is assumed that both the <external string> and <internal string> 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 string with leading nulls (4"00"). A null compared with any character is always considered a true comparison by the translate function.

An integer string of less than six characters is right justified with leading zeroes. An alpha string of less than six characters is left justified with trailing blanks.

If within a given function the length of each <internal string> is the same and the length of each <external string> is the same, it makes little difference whether the strings are UNEDITED, INTEGER, or ALPHA. However, if strings vary in length, using INTEGER or ALPHA strings can help to avoid confusion. For example, suppose a function is declared as follows:

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

Upon input, if the translate function were to search for an <external string> of 1, it would get a match with 11 because of a NNNN1. The source text after justification will compare as equal to NNNN11, the <external string> after justification (where N is a null). A similar phenomenon would occur on output if the translate function was searching for an <internal string> of ME: NNNNME would match NNSOME. This problem could be avoided by declaring the function as follows:

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

With this declaration, if the source text to be translated on input were "1", it would be converted to 00001. It would not match 000011, but would successfully match 1 justified as an integer. Likewise, source text on output of ME would be converted to MEBBB (where B is a blank); it would not match SOME BB, but would match ME justified as an alpha string.

FORMAT DECLARATION.
Syntax:

<format declaration>	::= FORMAT part list>.
<format part list>	::= <format part> / <format part list> , <format part>
<format part>	::= <format identifier> <special action part> <format description>
<format identifier>	::= <identifier>
<special action part>	::= [<special action>] / <empty>
<special action>	::= RESIDENT
<format description>	::= (<local declaration part> <editing specifications>)
<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>	::= a <integer> / <empty>
<editing specifications>	::= <editing phrase list>
<editing phrase list>	::= <editing phrase> / <editing phrase list> , <editing phrase>
<editing phrase>	::= <editing string> / <item phrase> / <location specifier>
<location specifier>	::= a <sign> <integer>
<sign>	::= + / - / <empty>
<editing string>	::= <simple string> / <skip field>

<p>GLOBAL Section cont</p>

```

<skip field> ::= X <integer> / X(<delimiter>)
<item phrase> ::= <repeat part> <item type>
                 <field width> /
                 <repeat part>
                 (<editing phrase list>) /
                 T(<function identifier> , <item type>
                 <field width> , <internal size>)

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

<update variable> ::= <variable identifier> : / <empty>
<item type> ::= A / I / B / J
<function designator> ::= <identifier>
<field width> ::= <integer> /
                 (<variable field specifier>)
<variable field specifier> ::= <delimiter> , <internal size>
<internal size> ::= <integer>
<delimiter> ::= <EBCDIC unit string> /
               <HEX unit string>
<simple string> ::= <EBCDIC code> <EBCDIC string> /
                  <hexadecimal code>
                  <hexadecimal string>
<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>"
<EBCDIC code> ::= 8 / <empty>
<hexadecimal code> ::= 4

```

Semantics:

The <format declaration> is used to define how a screen is to be built and/or how message text is to be modified. When formats are declared in the <format declaration>, the <DEVICE section> is used to indicate which formats are to be applied to which messages. Up to 1023 formats may be declared.

The <format part list> allows several formats, separated by commas, to be described in a single <format declaration>. Even though the syntax allows several formats to be described in one <format declaration>, it is good practice to define one format per <format declaration>. When a syntax error is encountered in a <format part>, the TCL scanner skips past any remaining <format parts> to the next <format declaration>. Syntax errors in the skipped <format parts> are not flagged until the <format part> in error is corrected. If one format is defined per <format declaration>, more syntax errors can be caught in each run of the TCL compiler.

Each <format part> associates a <format identifier> with a particular set of message formatting instructions. The <format identifier> is referenced in a <FORMATSIN statement> and/or a <FORMATSOUT statement> of the <DEVICE section>.

The <special action part>, if present, indicates whether the format is a resident format. A resident format is kept in an array in memory instead of on disk. This facility is provided for small, frequently used formats. It is intended to save the input/output overhead that would otherwise be required to retrieve a format from disk before using it. This option should be used with care since its overuse could require significant amounts of memory.

The <format description> consists of an optional <local declaration part> and the <editing specifications> enclosed within parentheses. Refer to <repeat part> under Editing Specifications for a discussion of the <local declaration part>. (Readers unfamiliar with GEMCOS formatting should refer to BASIC GEMCOS FORMATTING PRAGMATICS before continuing.)

The <editing specifications> describe the order and length of the fields of a message as well as the manipulation of the message buffer pointers. The <editing specifications> is a list of <editing phrases>. An <editing phrase> can be an <editing string>, a <location specifier> or an <item phrase>.

An <editing string> is either a <simple string> or a <skip field>. The <simple string> is used to place a literal field into a formatted message. A <simple string> can be an <EBCDIC string> such as "XYZ" or a <hexadecimal string> such as 4"0D" (carriage return). The <simple string> is used extensively when building forms for screen devices. It can be used to create the descriptive text of the protected areas as well as the necessary control characters. A <simple string> causes the pointer into the formatted message buffer to be updated to the right by the length of the string. The pointer into the source message buffer is unaffected by a <simple string> <editing phrase>.

Upon input the <skip field> causes text in the terminal message buffer to be skipped (by updating the terminal message buffer pointer). The number of characters skipped can be defined by an <integer> or a <delimiter>. For example, X3 causes three characters to be skipped while X(",") causes text up to and including the next comma encountered to be skipped.

Upon output, X8 causes eight spaces to be placed into the terminal message buffer while updating the pointer. X(<delimiter>) is undefined for output messages. The program message buffer pointer is unaffected by a <skip field>.

The <location specifier> is used to manipulate the program message-buffer pointer without affecting the terminal message-buffer pointer. By manipulating the program message-buffer pointer, fields can be skipped, re-ordered and/or re-used. There are two variations of the <location specifier> differentiated by the existence of an optional <sign>. When a <sign> is present, the program message-buffer pointer is adjusted by <integer> positions to the left (<sign> is a "-") or to the right (<sign> is a "+"). If there is no <sign>, the program message-buffer pointer is set to position <integer>. Care must be taken to keep the pointer within the bounds of the program message buffer. Upon input, the user should also be careful not to overlay good data in the program message buffer.

An <item phrase> defines a field of a formatted message. A field can be comparatively simple such as six alphanumeric characters, or rather complex, such as a repetition of several variable-length subfields. In order to encompass the wide variety of possible fields, several forms of the <item phrase> are available. All involve at least one <item type>, <field width> pair.

The <item type> determines how a field or subfield is to be edited. Four <item types> are available: A, B, I and J. A denotes an alphanumeric field, and B specifies a tabbed alphanumeric field. Alphanumeric fields may contain any characters, and leading blanks are considered significant. Truncation or blank filling occurs on the right. I denotes an integer field while J specifies a tabbed integer field. Integer fields may only contain digits and/or blanks except for imbedded blanks. They are truncated or right justified with zero filling on the left.

The <field width> determines the length of a field or subfield. Fields can be fixed or variable in length.

The simplest form of the <item phrase> is an alphanumeric or integer field with an <integer> <field width> such as A6 or I9. An A6 <item phrase> would result in the move of six characters from the buffer containing the raw message to the formatted message buffer. An <item phrase> of I9 would move nine characters subject to the editing rules already mentioned. The unprotected areas of formatted screens are usually composed of fixed alphanumeric or integer fields.

A more powerful form of the <item phrase> employs a <variable field specifier> <field width> such as A("+",6) or I("+",8). The <internal size> determines the size of the field in the program message buffer.

NOTE

While field lengths of the terminal message buffer may vary, field lengths of the program message buffer are always fixed. The <delimiter> is used to signify the end of the field in the terminal message buffer. The field begins where the previous field ends.

Upon input, a variable-length field is isolated based on the end of the last field and the <delimiter>. It is moved into the program message buffer justified according to the <item type>. The <delimiter> is not considered one of the characters of the field and, therefore, is not placed into the program message buffer.

Upon output, a string of characters of length <internal size> is obtained from the program message buffer. It is compressed by truncating trailing blanks or leading zeroes depending on the <item type>. The compressed string is placed into the terminal message buffer, and the <delimiter> is inserted after the compressed string.

During both input and output, the terminal message buffer is updated to the position following the <delimiter>, while the program message buffer is moved to the right by <internal size> positions.

Tabbed fields, where the <item type> is "B" or "J", are similar to variable-length fields on input and the same as fixed fields on output. Input, a tabbed field can end early if the tab character (4"05") is encountered. However, unlike a variable field, where the <delimiter> must be present, the tab character is not required to end the field. If enough characters are found, the field ends automatically. For example, a B10 <item phrase> on input causes characters to be moved from the terminal message buffer to the program message buffer until either ten characters have been moved or a tab character is encountered. The program message-buffer pointer is moved ten characters to the right. The terminal message-buffer pointer is left pointing to the eleventh character or to the character following the tab, whichever happens first. If the transfer is terminated by a tab character, trailing blanks are placed in the program message buffer to fill out all ten character positions. The tab character is not placed into the program message buffer.

Upon output, B5 would achieve exactly the same results as A5 and J7 would behave the same as I7. The tab character is not placed into the terminal message buffer as is done with the <delimiter> of a variable-length, nontabbed field.

The default tab character (4"05") can be changed by using a <variable field specifier> along with the B or J <item type>. J ("*",5) is the same as J5 except that "*" is the tab character instead of 4"05". B(4"05",10) is identical to B10.

Each <item phrase> discussed thus far may be repeated by placing a <repeat part> in front of the <item type>. A <repeat part> may be fixed or variable.

A fixed <repeat part> is designated by an <integer>. It is a shorthand method of representing an <editing phrase list> where each <editing phrase> is identical. For example, 2A6 is the same as A6,A6.

A variable <repeat part> can only be used on output. It is useful for messages which have a variable number of fields of repeated data such as tables with columns of values. These messages must have, as one of the data fields, a counter specifying the number of times a particular field will occur.

If a message is to contain a variable <repeat part>, the format applied to the message must have a <local declaration part>. The <local declaration part> specifies where in the message the counters governing the occurrence of the repeated fields are to be found. Values for variables declared are the first items extracted from the program message buffer. During each variable assignment, the program message-buffer pointer is adjusted by a combination of the <optional location specifier> and the length of the counter field. The length of the counter field is determined by the <integer> following the keyword FOR. The value of the counter contained in the program message buffer must be expressed as EBCDIC digits with a value not greater than 255. As many as six variables can be declared per format.

After a local variable has been set to a value extracted from the program message buffer, it can be referred to as a <variable identifier> in a variable <repeat part>. A variable <repeat part> consists of an optional <update variable>, a <variable identifier>, the keyword OR and an <integer>. The object of the <repeat part> is repeated either the number of times referred to by <variable identifier> or <integer> times, whichever is less. If the <update variable> is present, its variable identifier is set to <variable identifier> minus the number of times the repeat object was repeated. For example, V2 OR 8 would cause its object to be repeated V2 times, but not more than eight times. If V2 had a value of nine, V3:V2 OR 3A5 would cause A5 to be repeated three times and V3 would be set to 6. The original value of V3 is lost. If V2 had been zero, the A5 field would not occur and V3 would be set to zero.

An <editing phrase list> enclosed in parentheses is an even more complicated <item phrase>. This form can be thought of as a field composed of several subfields. An <editing phrase list> enclosed in parentheses can be the object of a <repeat part>. <Editing phrase lists> can be nested to 32 levels of parentheses.

Another complicated form of the <item phrase>, T(<function identifier>, <item type> <field width>, <internal size>), is a reference to a translate function. The <function identifier> refers to a function which must have been defined in a <function declaration>. The <item type> <field width> describes a field in the terminal message buffer, while <internal size> describes a field in the program message buffer.

FORMATTING ERRORS.

When an error is detected while formatting an input message, the MCS sets the format error field of the Common-area header to a nonzero value as described below. The message is then sent to the application program for which it was bound.

When an error is detected while formatting an output message, the MCS message is still sent to the destination station, but, in addition, an error message is sent to the control station specifying what type of error occurred.

<u>Error_Type</u>	<u>Description</u>
1	Destination pointer out of bounds
2	Source pointer out of bounds
3	Nondigit in integer field
4	Missing skip delimiter
5	Attempt to use variable repeat on input
6	Missing delimiter or variable field too long
7	Invalid string in translate field

Only the first error encountered is reported; however, the MCS attempts to continue formatting a bad message. When a type-3 formatting error occurs, the nondigit is placed into the erroneous field. For type-6 errors, significant text may be truncated in an attempt to force excessive data into the program message buffer. Type-7 errors result in question marks being placed into the erroneous field. Results are undefined for the other types of errors.

Tables 3-1 thru 3-5 list five graded examples (example sets 1 thru 5) of three increasingly difficult formats applied to input messages and output messages.

Example set 5 (table 3-5) uses the following function declarations:

```

FUNCTION GENDER(" MALE":"1","FEMALE":"2").
FUNCTION NUM1("ONE":"1","TWO":"2","THREE":"3","FOUR":"4",
             "FIVE":"5","SIX":"6","SEVEN":"7","EIGHT":"8",
             "NINE":"9","TEN":"10","ELEVEN":"11",
             "TWELVE":"12").
FUNCTION NUM2 [EXTERNAL:ALPHA,INTERNAL:INTEGER]
             ("ONE":"1","TWO":"2","THREE":"3","FOUR":"4",
             "FIVE":"5","SIX":"6","SEVEN":"7","EIGHT":"8",
             "NINE":"9","TEN":"10","ELEVEN":"11",
             "TWELVE":"12").
FUNCTION DAY ("1":"SUN","2":"MON","3":"TUE","4":"WED",
             "5":"THU","6":"FRI","7":"SAT").

```

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	ABC1234XY	A3,I4,A2	ABC1234XY
Input	ABC 4XY	A3,I4,A2	ABC0004XY
Input	ABC 4 XY	A3,I4,A2	ABC0004XY
Input/ Output	ABC0004XY	A3,I4,A2	ABC0004XY
Input	AB 5678XY	A3,X4,A2	AB XY
Input	AB GGGGX	A3,X4,A2	AB XY
Input/ Output	AB XY	A3,X4,A2	AB XY
Input	ABCDE	A2,"*",A3	AB*CDE
Input	AB*CDE	A2,"*",A3	AB**CD
Output	AB*CDE	A2,"*",A3	ABCDE
Input	RIGHT	A5,8"FACE"	RIGHTFACE
Output	RIGHTFACE	A5,8"FACE"	R IGH T
Output	NAME: [HARRY]C D 2	"NAME: [",A5,"]"",4"12"	HARRY
Output	Name: []C D 2	"NAME: [",A5,"]"",4"12"	(forms request)

Figure 3-4. Example Set 1 - Formatting Specifications Applied to Input and Output Messages

<u>Input/ Output</u>	<u>Message As It Appears at the Terminal</u>	<u><Editing specifications> Applied to Message In Transit</u>	<u>Message As It Appears to the User Program</u>
Input	1234XY	I6	1234XY (FMterr set to 3)
Output	1234XY (control station notified of error)	I6	1234XY
Input/ Output	ABCDXYZ	24, A4, 21, A3	XYZABCD
Input/ Output	ALPHA	23, A5	ALPHA
Input/ Output	AB123XY456	A2, 25, I3, 23, A2, 28, I3	ABXY123456
Input/ Output	ABCD	2A2	ABCD
Input/ Output	AB12CD34	2(A2, I2)	AB12CD34
Output	01/28/52	I2, 2(""/", I2)	012852
Input	01/28/52	I2, 2(X1, I2)	012852
Output	* AB CD EF	Variable V1 for 2; "*/", V1 or 5(X2, A2)	03ABCDEF
Output	XX 1 2 3YY 4 5	Variable V1 27 for 2, V2 25 FOR 2; 21, A2, 29, V2: V1 or 3(X1, I1), 23, A2, 212, V2 or 3(X1, I1)	XXYY000512345

Figure 3-5. Example Set 2 - Formatting 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	E*	A("*",3)	E
Input/ Output	E*15P	A("*",3), I("P",5)	E 00015
Input	ABCDEFGG+	A("+",4)	ABCD (FMterr set to 6)
Input	12345*AB	I("*",4), A2	1234AB (FMterr set to 6)
Input	T A1B2AXYZ B	86,B3	A1B2 XYZ
Input	A1B2C3XYZ	86,B3	A1B2C3XYZ
Input	T A1B2C3AXYZ B	86,B3	A1B2C3
Input	T T 12A34A B B	2J5	0001200034
Input	T 123456A B	2J5	1234500006
Input	TT AA BB	2J5	0000000000

Figure 3-6. Example Set 3 - Formatting Specifications Applied to Input and Output Messages

GLOBAL Section
cont

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	A1B2 XYZ	B6,B3	A1B2 XYZ
Input/ Output	0001200034	2J5	0001200034
Input/ Output	ABCDEF123456	B(" ",6),J(" ",6)	ABCDEF123456
Input	A*1+	B(" ",6),J(" ",6)	A 000001
Input/ Output	A 000001	B(" ",6),J(" ",6)	A 000001
Input/ Output	ABCDEF123456	3(A2,2+2),23,3(I2,2+2)	AB12CD34EF56
Input/ Output	IJGHEFCDA8	29,4(A2,2-4),A2	ABCDEFGHIJ
Input	XYZ12345,ABC	A3,X(" ",)A3	XYZABC
Input	XYZ,ABC	A3,X(" ",)A3	XYZABC
Input	XYZ ,ABC	A3,X(" ",)A3	XYZABC
Input	XYZABC	A3,X(" ",)A3	XYZ (FMTERR set to 4)
Output	ABCDEFGHI	"ABC","DEF","GHI"	QRST123
Output	D CRESULTS=0053 4	4"OC","RESULTS=",I4	0053
Output	D CRESULTS=0000 4	4"OC","RESULTS=",I4	(forms request)

Figure 3-7. Example Set 4 - Formatting Specifications Applied to Input and Output Messages

<u>Input/ Output</u>	<u>Message As It Appears at the Terminal</u>	<u><Editing specifications> Applied to Message In Transit</u>	<u>Message As It Appears to the User Program</u>
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	TW03	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 (FMterr set to 7)
Output	(FOUR)	"(",T(NUM2,A(" ",6),2)	04
Input	WED	T(DAY,A3,1)	? (FMterr set to 7)
Output	??? (control station notified of error)	T(DAY,A3,1)	2
Input/ Output	3	T(DAY,A1,3)	WED
Input/ Output	ONE X	T(NUM2,A6,1),A1	1X

Figure 3-8. Example Set 5 - Formatting Specifications Applied to Input and Output Messages

BASIC GEMCDS FORMATTING PRAGMATICS. This discussion attempts to explain the basic concepts of formatting. It should prove helpful to the user who has not yet worked with GEMCDS formatting.

The MCS uses two buffers when formatting a message: one buffer contains the message as it appears at the terminal; the other contains the message as it appears to the Application Program. A message consists of a sequence of one or more fields just as a disk, tape or card record is composed of a sequence of fields. A format describes the relationship between the fields of a message that are written/read by a program and the fields of the message that are received/transmitted by a terminal.

Input formatting causes a message in the terminal message buffer to be moved, field by field, to the program message buffer. Output formatting moves fields from the program message buffer to the terminal message buffer. When a field is moved, whether by input or output formatting, it is moved under the control of an <item phrase>, the terminal message-buffer pointer and the program message-buffer pointer.

An <item phrase> consists of a field type, a field length and an optional field delimiter. The field type defines which characters are valid in a field and controls its justification and fill. The field length determines the number of characters in the field. The field delimiter, if present, designates the character which ends a field. The Terminal message-buffer Pointer (PT) refers to a particular character position in the terminal message buffer. Likewise, the Program message-buffer Pointer (PP) refers to a particular character position in the program message buffer.

Pointers PT and PP both begin pointing at the first character (position 1) in their respective messages. As the editing phrases of a format are applied to data fields, the data is moved from one message buffer to the other, and the pointers are updated. Unless specifically instructed to do otherwise, the pointers are updated by moving to the right by the number of characters moved.

Example:

Assume that the message "ABC 123" was received from a terminal and it was determined that the format (A3,I4) was to be applied. The situation would initially appear as depicted in figure 3-9, with the message placed in the terminal message buffer, and the program message buffer cleared and the pointers initialized. The A3 <item phrase> controls the move of the first three alphanumeric characters as depicted in figure 3-10. As can be seen, "ABC" is placed into the program message buffer, and the pointers are moved three positions to the right.

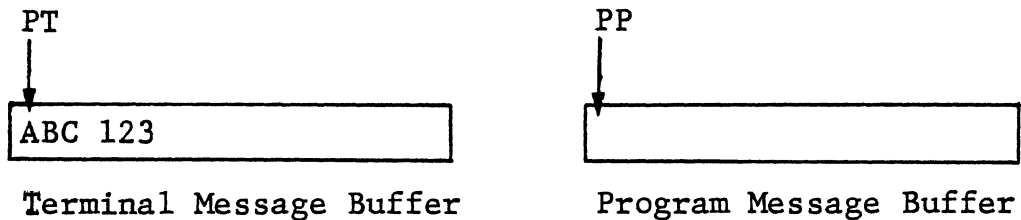


Figure 3-9. Initial Contents of Terminal and Program Message Buffers

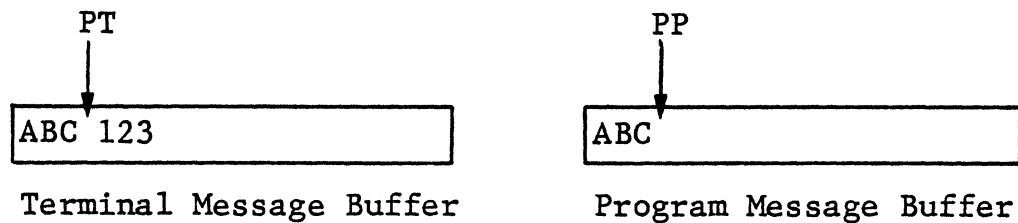


Figure 3-10. Contents of Terminal/Message Buffers After Move Caused by A3 <item phrase>

Then, the I4 <item phrase> causes " 123" to be moved. During output, integer fields are right justified with zeroes filled and/or blanks converted to zeroes. This "0123" is placed into the program message buffer. Figure 3-11 shows the final situation. At this point, the program message buffer is sent to the appropriate Application Program.

A higher degree of formatting flexibility may be achieved by moving the pointers without moving text. The Terminal message-buffer Pointer (PT) may be advanced without affecting the Program message-buffer pointer (PC) by using a <skip field> (i.e., the X <editing phrase>); but only PT may be advanced. PP may be moved in either direction without affecting PT by using a <location specifier>.

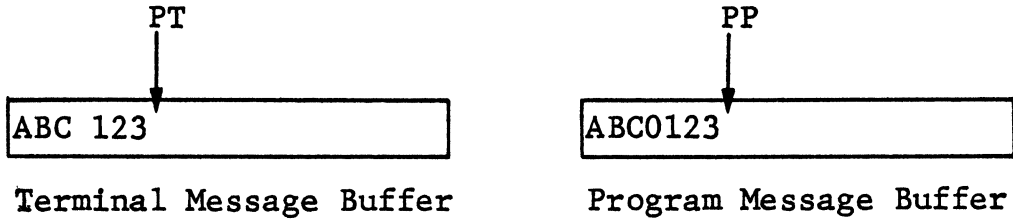


Figure 3-11. Contents of Terminal/Message Buffers After Move Caused by I4 <item phrase>

Figures 3-12 thru 3-18 illustrate the effect of applying the formats (A2,24,A1,X1,23,A1) to the output message "WXYZ" using the <skip field> and the <location specifier>.

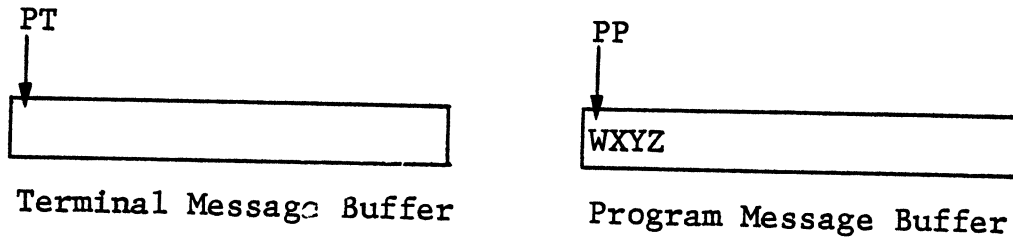
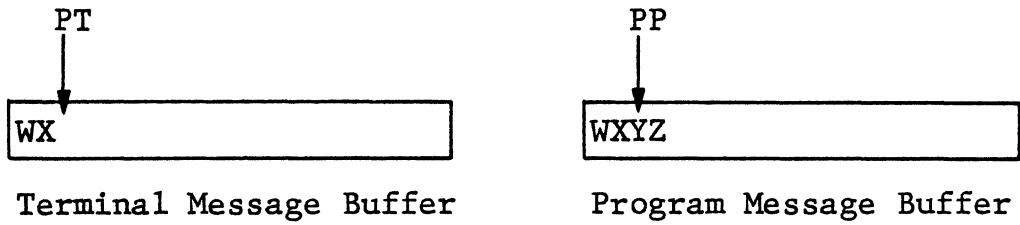
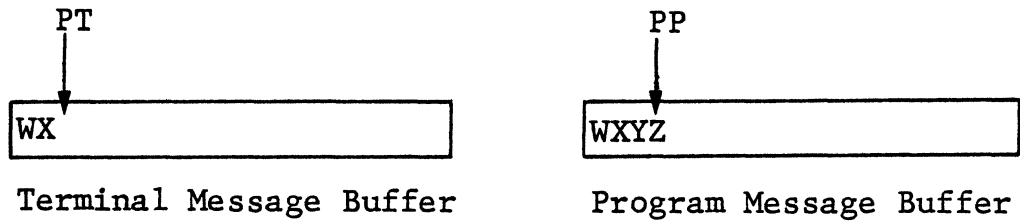


Figure 3-12. Contents of Initialized Buffers



**Figure 3-13. Buffer/Pointer Update
After Applying
Specification A2**



**Figure 3-14. Buffer/Pointer Update
After Applying
Specification A4**

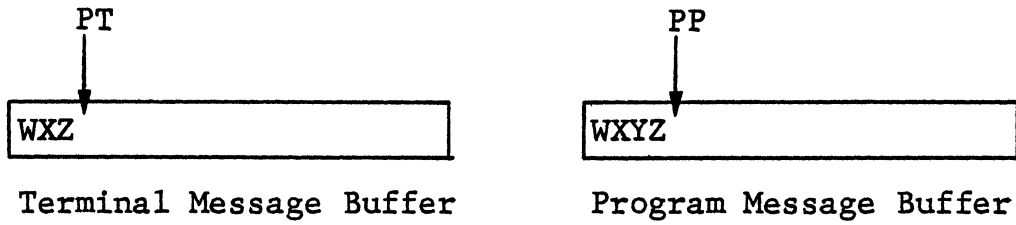


Figure 3-15. Buffer/Pointer Update
After Applying
Specification A1

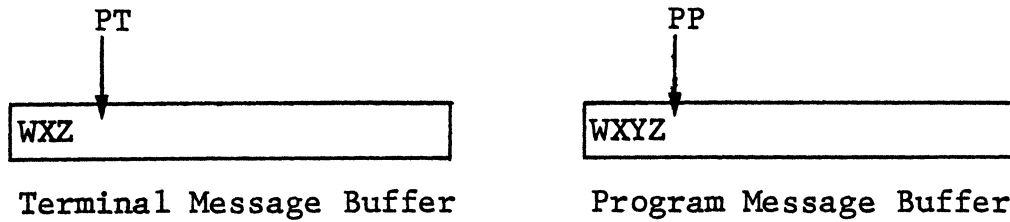


Figure 3-16. Buffer/Pointer Update
After Applying
Specification X1

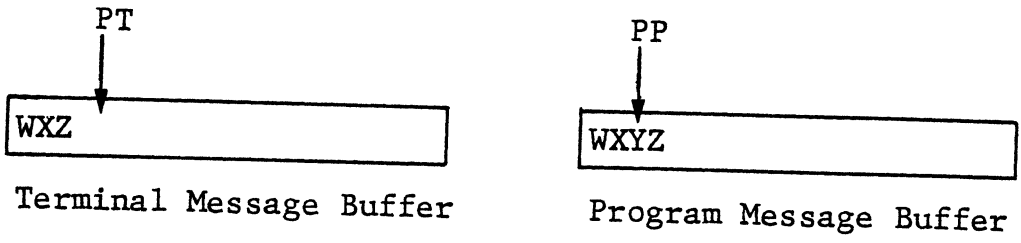


Figure 3-17. Buffer/Pointer Update
After Applying
Specification 23

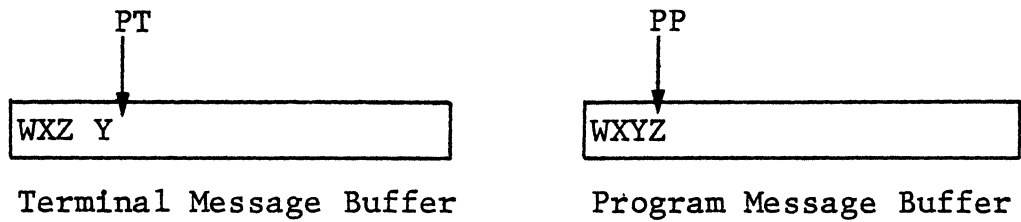


Figure 3-18. Buffer/Pointer Updates After
Applying Specification A1
and Sending the Terminal
Message Buffer Contents

RECALL PROGRAM STATEMENT.

Syntax:

```
<RECALL PROGRAM statement> ::= RECALLPROGRAM = <identifier>./  
                                <empty>
```

Semantics:

The <RECALL PROGRAM statement> specifies which program is to be designated as the Recall Program. The Recall Program is used to recall both audited input and output messages. See section 8 for further explanation of the Recall Program. GENCOS supplies a Recall Program called MCSRECALL on the release tape. Identifier must be 10 characters or less in length. By default, there is no Recall Program.

Examples:

```
RECALLPROGRAM = MCSRECALL.  
RECALLPROGRAM = RECALLPROG.
```

CONTROL STATIONS STATEMENT.

Syntax:

```

<CONTROLSTATIONS statement> ::= CONTROLSTATIONS =
                                <station identifier>. / <empty>

```

Semantics:

The <CONTROLSTATIONS statement> allows one station to be designated as a Control station. Privileged Network Control Commands may only be entered at the system console, the card reader or the Control station. Errors monitored by the MCS are reported to the Control station if one is specified (otherwise the system console is used). A <CONTROLSTATIONS statement> cannot occur in a regenerate MCSTCL run if it did not occur in the GENERATE run. If the <CONTROLSTATIONS statement> occurs in the GENERATE run, its value may be changed in a REGENERATE run. The <station identifier> must appear as a <station name> in the <STATION section>. By default, no Control station is specified and no supporting logic generated.

Examples:

```

CONTROLSTATIONS = TD800A.
CONTROLSTATIONS = MANAGER.

```


DEFINITION Section

DEFINITION_SECTION.**Syntax:**

```
<DEFINITION section> ::=
    BEGIN
    <ACCESS CONTROL statement>
    <PROGRAM section>
    <STATION section>
    <DEVICE section>
    <MESS CODE section>
    END.
```

Semantics:

In the <DEFINITION section>, the user defines access keys (user IDs), programs and stations as well as their interrelationships. If the user requires MCS functions not supported by GEMCOS, UPL source code statements can be merged into a GEMCOS MCS by including a <MESS CODE section> in the <DEFINITION section>.

ACCESS CONTROL STATEMENT.

Syntax:

```

<ACCESS CONTROL statement> ::= ACCESSCONTROL :
                                <association list> / <empty>

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

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

<access code>                 ::= <identifier>

<item list>                   ::= <item> / <item> , <item list> /
                                ALL

<item>                        ::= <trancode> / <program name>

```

Semantics:

The <ACCESS CONTROL statement> allows for the specification of access codes. An access code is required as part of the sign-on command syntax (*SGN access code) and identifies the user signing on to the MCS. An access code identifier is an alphanumeric identifier up to six characters in length. Associated with each access code is an <item list> consisting of transaction codes (trancodes) and/or <program names> which that particular user is authorized to use.

DEFINITION Section
cont

When a message is received from a station, the MCS searches for a transaction code in the message. If so, the MCS determines if the access code used to sign-on at that station is authorized to use that tranocode. If the access code is authorized, the message is routed to the appropriate program; otherwise, an error is returned to the station. If a tranocode could not be found in the message, the MCS verifies that the access code is authorized to use the program currently attached to the station. If so, the message is routed; if not, an error is reported.

NOTE

If the value of sign-on for a station is FALSE, access control is not in affect at that station. No messages entered at such a station are rejected due to access control restrictions.

Each tranocode encountered in the <ACCESS CONTROL statement> must appear in a <TRANCODE statement> of the <PROGRAM section>. Likewise, each <program name> must appear in a <program define> of the <PROGRAM section>. If a signed-on user is to have unrestricted use of all the defined transaction codes and programs, the key word ALL may be used. If ALL is used, it must be the only <item> in the <item list>.

Example:

```
ACCESSCONTROL :  
  ACCESSKEY ABCD   = INQ, PAYROLL.  
  ACCESSKEY AB1234 = ALL.  
  ACCESSKEY AB5678 = INQ, XYZ.
```

PROGRAM_SECTION.

Syntax:

```

<PROGRAM section> ::= <PROGRAM DEFINE list>

<program define list> ::= PROGRAM DEFINE> /
                        <program define list>
                        <program define>

<program define> ::= PROGRAM <program name>
                  <program classification> :
                  <program description>

<program name> ::= <identifier>

<program classification> ::= ASSIGNMENT / UTILITY / USER /
                          <empty>

<program description> ::= <PROGRAM STATEMENT list>

<PROGRAM STATEMENT list> ::= <PROGRAM statement> /
                             <PROGRAM STATEMENT list>
                             <PROGRAM statement>

<PROGRAM statement> ::= <INTERFACE statement> /
                       <TRANCODE statement> /
                       <PROGRAM TITLE statement> /
                       <RESIDENCE statement> /
                       <COMMON SIZE statement> /
                       <EXECUTE statement> /
                       <RECOVERY statement> /
                       <DATABASE NAME statement> /
                       <AUDIT TRANSACTIONS statement> /
                       <AUDIT ASSIGNMENT statement> /
                       <AUDIT OUTPUT statement> /
                       <RESTART PROGRAM statement> /
                       <MAXCOPIES statement> /
                       <OPEN MESSAGE statement> /
                       <ATTACH MESSAGE statement> /
                       <DETACH MESSAGE statement> /
                       <CONVERSATIONSIZE statement> /
                       <MAXASSIGNERS statement> /
                       <AP300STATUS statement> /
                       <TRANSACTION CODE POSITION statement>

```

**PROGRAM Section
cont**

Semantics:

The library of on-line programs is defined in the <PROGRAM section>. All programs that open a remote file which is to be approved by the GEMCDS MCS must appear in the <PROGRAM section>. If a program attempts to open a remote file consisting of at least one station in the GEMCDS MCS remote file (identified by the <QUEUE NAME statement> of the <GLOBAL section>), and if the program does not appear in the <PROGRAM section>, the MCS does not allow the file to open.

The <PROGRAM section> is composed of a <program define list>. Each <program define> specifies a <program name>, a <program classification>, and a <program statement list>.

The <program name> is any alphanumeric identifier. If there is an <ACCESS CONTROL statement>, <program name> may appear in its <item list> to allow certain <access codes> to use the program.

The <program classification> specifies to the MCS how this program can be executed as well as how messages are to be routed to it once it is running. As of the 3.0 GEMCDS release, there are three <program classifications>: ASSIGNMENT, UTILITY and USER. By default, <program classification> is ASSIGNMENT.

ASSIGNMENT PROGRAMS.

An Assignment Program may only be executed from the supervisory console, a card reader or the Control station. An attempt to execute an Assignment Program from any other than the Control station by means of the EX Network Control Command results in an operator error.

After being executed, an Assignment Program eventually opens a remote file in order to gain control of a list of stations in the network. A GEMCDS MCS grants control of a particular station to an Assignment Program if the MCS controls the station, and if no other Assignment or Utility Program controls the station. The MCS controls a station if that station appears in the remote file opened by the GEMCDS MCS (refer to stations controlled by a GEMCDS MCS in section 2). When an Assignment Program opens a remote file, the MCS checks each station defined to be in the program remote file. If the MCS determines that it cannot grant control of any of these stations, the FILE OPEN is denied. Otherwise, the MCS approves the file open request for the stations in the list for which it is able to grant control. Once control of a station is given to an Assignment Program, all messages entered from that station that do not contain a truncate code of a User Program are routed to the Assignment Program (assuming access control is not violated).

An Assignment Program retains control of its stations until it resolves to close its remote file. If a HAP Network Control Command is entered from the Control station, the supervisory console or a card reader, the MCS places an End-of-File character into the queue of the Assignment Program, which prompts it to close its remote file and go to End-of-Job. When an Assignment Program closes its remote file, the stations are no longer considered busy and can be attached to another Assignment or Utility Program.

Thus, the GEMCDS MCS handles file opening and message routing for an Assignment Program in much the same way that a Network Controller does when no MCS is present. However, GEMCDS also provides an Assignment Program with additional functions such as a Common-area header, truncate indicies, access control, audit, recovery, and formatting.

UTILITY PROGRAMS.

A Utility Program may only be executed from a station in the network. An attempt to use the EX Network Control Command to execute a Utility Program from the supervisory console or a card reader is denied. A station may not "EX" a Utility Program if that station is already controlled by an Assignment Program or another Utility Program since the station would be considered busy.

Upon receipt of an EX network control command from the station, the MCS determines, in the order listed, the status of the following as they pertain to the utility program:

- a. Program is running.
- b. Number of stations attached to the program exceeds the limit assigned.
- c. Number of program copies exceeds the limit assigned.

If the program is not running, the MCS initiates the program with the ZIP EXECUTE command. Afterward, the initiated program opens a dummy file. Afterward, the MCS attaches the requesting station. (For further information about dummy files, refer to Burroughs B 1700 Systems Network Definition Language Reference Manual, form number 1073715.)

If the program is running, the MCS checks whether the number of stations attached to this program exceeds the maximum assignment limit; if it does not, the MCS dynamically attaches the station to the remote file of the program. However, if the number of stations attached to the program does exceed the limit, the MCS then proceeds to check whether the number of program copies exceeds the limit established. If it does not, the MCS initiates a copy of the program and attaches the station to it. However, if the program copy limit is exceeded, the MCS displays an error message.

PROGRAM Section
cont

Once the attachment occurs, the utility program controls the station. All messages entered from that station which do not contain a trancode or a user program are routed to the utility program.

When the user is finished with a program, the HAP network control command is entered. This prompts the MCS to detach the station from the remote file of the utility program. The station is available and can be attached to another Assignment or Utility Program. If only one station was attached to the program copy, the MCS places an End-of-File character in the Utility Program queue (for that copy only). The character prompts the program to close the remote file and proceed to End-of-Job.

GEMCOS handles a Utility Program in much the same manner as the B 1700 illustrative MCS handles a program that opens a remote file. However, GEMCOS also provides a Utility Program with additional functions such as a Common-area header, trancode indicies, access control, audit, recovery, and formatting.

USER PROGRAMS.

A User Program, like an Assignment Program, may only be executed from the supervisory console, a card reader or the Control station. An attempt to execute a User Program from any station in the network other than the Control station by means of the EX Network Control Command is denied. A User Program must use an interface of PARTICIPATION.

After being executed, a User Program should open a remote file for stations it can service. The MCS approves the REMOTE FILE OPEN as long as the stations in the remote file are controlled by GEMCOS (those stations not in the remote file of the MCS being deleted from the remote file of the User Program).

NOTE

The MCS does not check to see if another on-line program controls the stations, since a User Program does not control stations.

Unlike an Assignment Program or Utility Program, a User Program receives a message entered from a station in its remote file only if the message has a trancode. At a given point in time, a station may be attached to as many User Programs as necessary since the MCS is able to switch messages entered at the station based on a trancode found in the message (a station may only be attached to one Assignment or Utility Program at a time and all messages without a trancode go to that program). A station may be simultaneously attached to an Assignment or Utility Program, even though it may still be attached to User Programs.

A User Program must have at least one <TRANCODE statement> in its <PROGRAM statement list>; otherwise, the program cannot receive any messages.

If several copies of a particular User Program are executed, the MCS distributes the message load evenly among them. This feature can increase system throughput since inputs/outputs (I/Os) can be overlapped.

A User Program continues to service the stations in its remote file until it closes its remote file. If a HAP Network Control Command is entered for this program, the MCS places an End-of-File character in the User Program queue, prompting it to go to End-of-Job.

Examples:

PROGRAM A ASSIGNMENT:
TITLE = PACKA/PAYROLL/.
TRANCODE = UPDATE.
COMMONSIZE = 60.

PROGRAM B UTILITY:
TITLE = EDIT/IT.
COMMONSIZE = 75.
RESIDENCE = CORE.

PROGRAM C USER:
TITLE = FIXIT.
TRANCODE = OLD(8,1).
TRANCODE = NEW(9,1).
RESIDENCE = DISK.

INTERFACE STATEMENT.

Syntax:

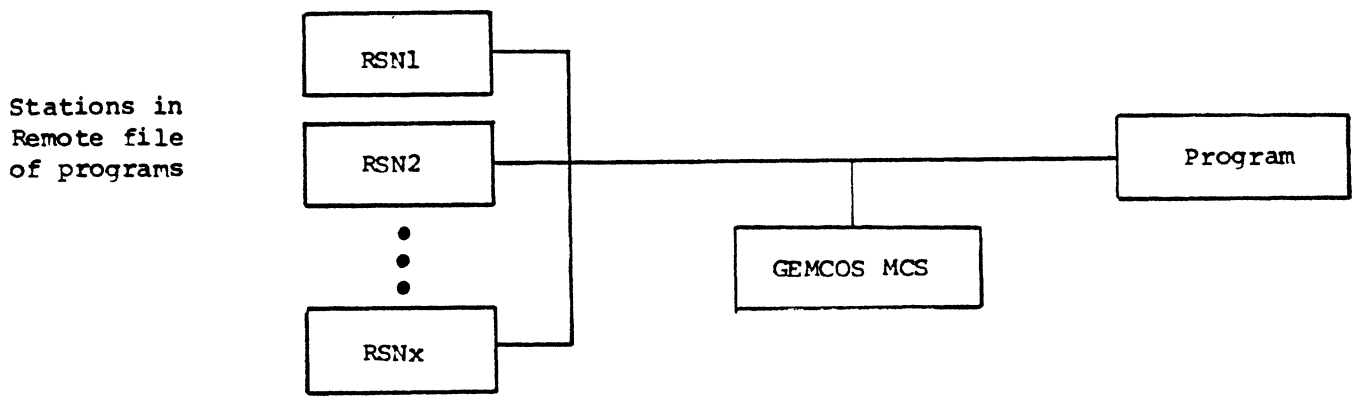
```
<INTERFACE statement> ::= INTERFACE = <program interface>. /  
                           <empty>
```

```
<program interface> ::= NONPARTICIPATION / PARTICIPATION / MCS
```

Semantics:

The <INTERFACE statement> determines the path messages follow as they flow between a particular program and the stations in its remote file. It also determines the relationship between the GEMCOS MCS and the program. Three interfaces are available: NONPARTICIPATION, PARTICIPATION and MCS. The NONPARTICIPATION and PARTICIPATION interfaces may only be used by application programs, programs which open a remote file without headers. The MCS interface may only be used by MCS programs, programs which open a remote file with headers. By default, interface is PARTICIPATION.

NONPARTICIPATION. A NONPARTICIPATION interface is an efficient but static method for a program to communicate with the stations in its remote file. Figure 3-19 depicts the flow of messages in a NONPARTICIPATION interface.



RSN signifies the Relative Station Number.

Figure 3-19. NONPARTICIPATION Interface

With NONPARTICIPATION interface all messages (except those beginning with a signal character) that are entered from all stations in the application program remote file go to the program. The program can write messages to any of its stations. A construct known as a remote key allows the program to determine the source and length of an input and to specify the destination and length of an output.

Messages written by the program or entered from a station beginning with a signal character are sent to the MCS. GENCOS Network Control Commands reach the MCS by means of this signal character when a NONPARTICIPATION interface is chosen.

Messages, beginning with two signal characters, that are entered from a station are processed by the MCS in the following manner:

- a. If a trancode is found in the message, the transaction is routed to the program specified by the trancode provided the program is running or declared as ONDEMAND. Output messages from the program are routed back to the station. This allows a user at a station, that is attached to a non-participating program, to perform trancode routing to other programs in the network.
- b. If the message (starting from the third character position only) contains a message-ID, it is considered to be a forms request, and the blank form is sent back to the station. This feature is only available in the advanced and total versions of GENCOS.
- c. If the message contains neither a valid trancode nor message-ID, it is routed to the program to which the station is attached. The first two bytes (or two signal characters) are not returned with the message.

The NONPARTICIPATION interface is efficient since a typical transaction passes through only one program, the User Program (in addition to the Network Controller). This interface is static since a station can only be in one opened (input) remote file at a time and therefore has access to only one program. In addition, the MCS does not have access to the normal flow of messages and is unable to provide audit, formatting, access control, and its other functions.

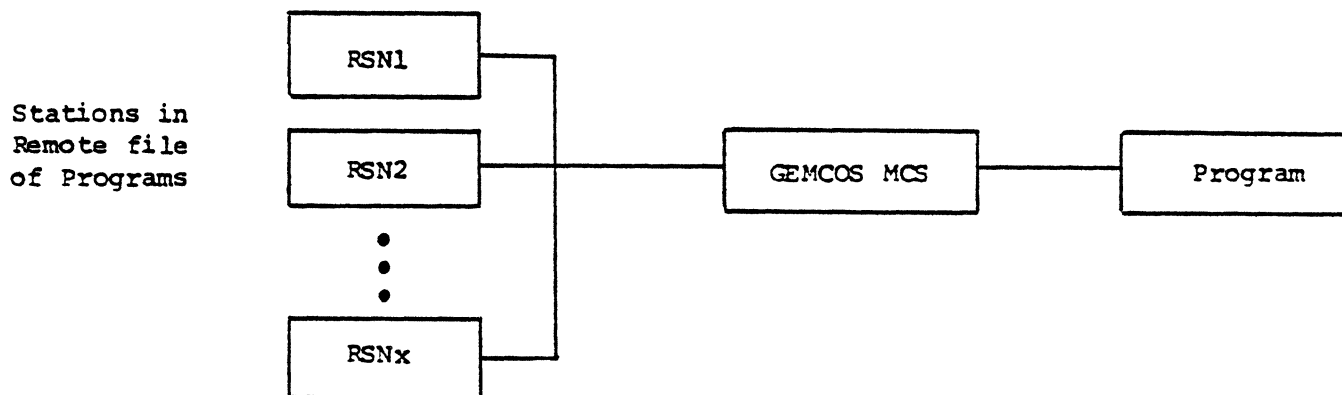
When interface is NONPARTICIPATION, the program classification cannot be USER and there cannot be any transaction codes. The Common-area header will not be on messages received by the program, and the program must not provide them on output. Thus, COMMONSIZE cannot be set. ATTACHMESSAGE, DETACHMESSAGE and OPENMESSAGE cannot be TRUE. Users at stations in the remote file of a Nonparticipation Program can neither use transaction-based routing nor initiate screen requests while the Nonparticipation Program is running. Even if station has been assigned a SCREENSIZE, screen wraparound cannot take place while the station is under control of a Nonparticipation Program. Audit, queue restoration and formatting are not possible (even though these options can be specified in the <GLOBAL section> and can be used by Participa-

PROGRAM Section
cont

tion Programs attached to other stations in the network while Nonparticipation Programs are running).

If stations can be dedicated to a particular program while the program is running and the program does not require access control, audit, queue restoration, formatting or screen wraparound, it is advantageous to use the Nonparticipation interface.

PARTICIPATION. When PARTICIPATION is specified as the program interface, all messages entered at stations pass through the MCS before being sent to programs, and all messages written by the program pass through the MCS before being transmitted to stations. The MCS is said to be "participating" in the message traffic flowing between the program and the stations of the program remote file. Figure 3-17 depicts the flow of messages in a PARTICIPATION interface.



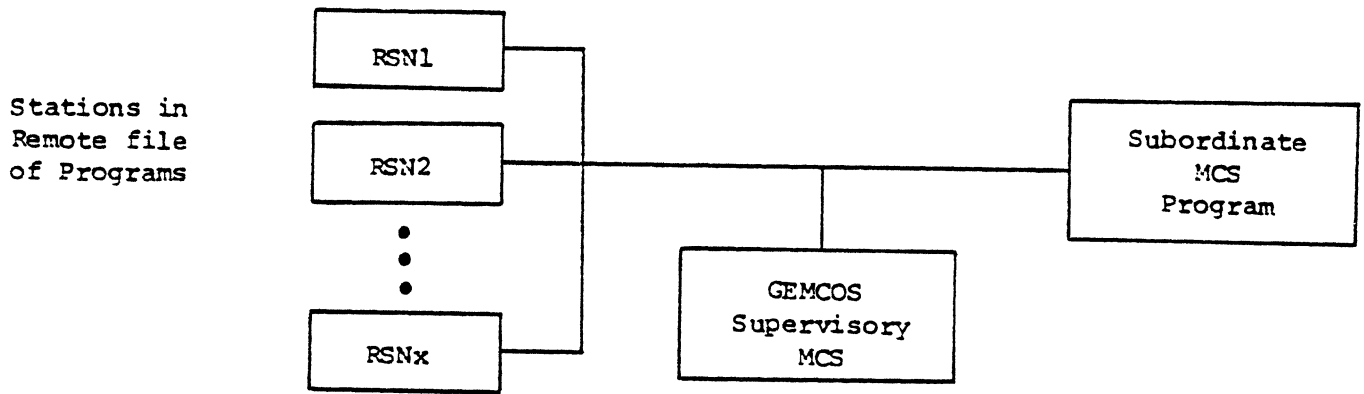
RSN signifies the Relative Station Number.

Figure 3-20. PARTICIPATION Interface

The PARTICIPATION interface is slightly less efficient in terms of throughput, since a typical transaction passes through three programs: the MCS (during input), the User Program, and the MCS again (during output). The slight decrease in efficiency is more than offset by the full complement of centralized functions provided by the MCS message. It can provide a full array of centralized functions (including audit, recovery, formatting, screen wraparound, access control, and various forms of routing).

Programs which use the PARTICIPATION interface receive and must provide the Common-area header. This header, in addition to its other functions, allows the program and MCS to communicate the message text length and the source/destination station to each other.

MCS. When a program is defined as using an MCS interface, the flow of messages is similar to a NONPARTICIPATION interface. All messages (except those beginning with the signal character) entered from each station in the MCS program remote file go to the program. The MCS program can write to any station in its remote file. Figure 3-21 depicts the flow of messages in an MCS interface.



RSN signifies the Relative Station Number.

Figure 3-21. MCS Interface

**PROGRAM Section
cont**

Two areas in which the MCS interface differs from the nonparticipation interface are as follows:

- a. A program using an MCS interface must open its remote file with headers, thereby identifying itself as an MCS to GEMCOS and the Network Controller.
- b. The program must provide and expect a Network Controller-defined 50-byte header preceding all data messages. With this header, the program may access tallies and toggles and may perform functions such as output message switching, communication with the Network Controller, remote file management, system interrogation, and system control. (The Network Controller/Message Control System interface is defined in Burroughs B 1700 Systems Network Definition Language Reference Manual, form 1073715.)

When a program using the MCS interface opens its remote file, GEMCOS assumes the status of a supervisory MCS while the program is considered a subordinate MCS. The supervisory MCS must be entered into the mix before any of the subordinate MCS programs.

A program using the MCS interface can be either a Utility Program or an Assignment Program. In brief, stations can dynamically attach to and detach from a Utility Program via the EX and HAP Network Control Commands, while an Assignment Program controls a fixed set of stations and can only be initiated from the Control stations, the supervisory printer or a card reader.

Messages entered at stations in a remote file opened by a subordinate MCS (which do not begin with the GEMCDS signal character) go directly to the subordinate MCS and are not seen by GEMCDS. As a result, an MCS program temporarily suspends GEMCDS MCS functions (except certain Network Control Commands) at the stations in its remote file. While stations are in the remote file of an MCS program, they cannot use GEMCDS trancodes, screen wraparound, audit, recovery or formatting. Messages beginning with the GEMCDS signal character go to the GEMCDS supervisory MCS so that, even while a station is attached to a subordinate MCS, GEMCDS network control commands can be entered.

NOTE

Network Control Commands affecting the attributes of stations in the remote file of an MCS program cannot be acted upon. The subordinate MCS is responsible for the attributes of the stations it controls.

The B 1800/B 1700 MCS/Network Controller interface allows subordinate MCS programs to change data communication attributes of associated stations. However, if a station attribute is changed by a subordinate MCS, the change is effective only while the subordinate MCS controls the station. As soon as either the subordinate MCS closes its remote file or the station detaches itself, GEMCDS returns the station to its original status.

Examples:

INTERFACE = NONPARTICIPATION.
INTERFACE = PARTICIPATION.
INTERFACE = MCS.

TRANCODE STATEMENT.

Syntax:

```
<TRANCODE statement> ::= TRANCODE = <trancode list>./  
                        <empty>  
  
<trancode list>      ::= <trancode> <trancode indices>/  
                        <trancode list>,  
                        <trancode> <trancode indices>
```

Semantics:

The <TRANCODE statement> is used to define trancodes and to associate them with programs. A trancode identifier is any string up to ten characters in length. A program of any classification which uses an interface of PARTICIPATION may have associated trancodes. However, only trancodes associated with User Programs cause transaction-based routing to occur. A trancode defined in a <TRANCODE statement> may occur in the <ACCESS CONTROL statement> to restrict its use to a specific list of <access keys>.

The <module-function indicies> may optionally be associated with each trancode. The <module-function indicies> consist of two <integer> values. Each <integer> may be a value from 0 to 63. If a trancode has <module-function indicies>, they are placed into the Common-area header of messages in which that trancode is present. The receiving program can use the <module-function indicies> in a UPL CASE statement or a COBOL GO TO DEPENDING ON in order to branch to the code which will process that trancode. This eliminates the need for the application program to determine which trancode has just been received. If a trancode has no <module-function indicies> or if there is no trancode in a message, zeroes are placed into the <module-function indicies> field of the Common-area header.

A User Program must have at least one <TRANCODE statement> in its <PROGRAM statement list>; otherwise, the program never receives any messages. The <TRANCODE statement> is optional in the <PROGRAM STATEMENT list> of Assignment and Utility Programs.

NOTE

If input formatting is to take place, a message must have a trancode regardless of the classification of the destination program, so that the MCS is able to determine which format is to be applied. The trancode is considered as one of the fields of a formatted message.

Examples:

```
TRANCODE = INQ (8, 10), UPDATE (5, 3).  
TRANCODE = FIX (18, 1).  
TRANCODE = HELP.
```

PROGRAM TITLE STATEMENT.

Syntax:

<PROGRAM TITLE statement> ::= TITLE = <file-ID>./<empty>

Semantics:

The <PROGRAM TITLE statement> identifies the object-code file name of a program. The <file-ID> is a B 1700 file identifier and is used in the EX, HAP, RPS and RPC Network Control Commands to refer to the program. The <PROGRAM TITLE statement> is optional. If it is omitted, the program title defaults to the first 10 characters of <program name>.

Examples:

TITLE = PACK1/X/Y.

TITLE = PGM1.

TITLE = A/B/.

<p>PROGRAM Section cont</p>

RESIDENCE STATEMENT.

Syntax:

```
<RESIDENCE statement> ::= RESIDENCE = <run-time residence>. /  
                           <empty>  
<run-time residence> ::= DISK / CORE
```

Semantics:

The <RESIDENCE statement> allows the user to specify where the program is to reside when not processing messages. The value of RESIDENCE may be CORE or DISK. A CORE Resident Program gives faster response but increases the memory requirements of the data communication system. The <RESIDENCE statement> is optional and defaults to CORE.

Examples:

```
RESIDENCE = CORE.  
RESIDENCE = DISK.
```

COMMON SIZE STATEMENT.

Syntax:

<COMMON SIZE statement> ::= COMMONSIZE = <integer>. / <empty>

Semantics:

The <COMMON SIZE statement> allows the user to specify the length of the header preceding the text of messages exchanged between the MCS and application programs using the PARTICIPATION interface. <Integer> must be a value from 60 to 200. Bytes 1 through 60 are reserved for GEMCOS-defined fields. Bytes 61 through 200 can be reserved for user-defined fields. User-written code must be merged into the MCS if it is to access, set, or modify bytes 61 through <integer>. The <COMMON SIZE statement> is optional. By default, COMMONSIZE defaults to 60 (no room reserved for user-defined fields).

Programs using either the NONPARTICIPATION or MCS interface cannot receive a Common-area, and thus COMMONSIZE cannot be set.

Examples:

```
COMMONSIZE = 60.  
COMMONSIZE = 200.
```

EXECUTE STATEMENT.

Syntax:

<EXECUTE statement> ::= EXECUTE = <execute option list>. / <empty>

<execute option list> ::= <execute list> /
<execute list>, <execute option list>

<execute list> ::= ONDEMAND / BOJ / MANUAL

Semantics:

The <EXECUTE statement> allows the user to reduce intervention by the Console or Control Station operator during program fire up. Three options are available: ONDEMAND, BOJ, and MANUAL. ONDEMAND and MANUAL may not appear together in the same <EXECUTE statement>. The default for this statement is MANUAL.

ONDEMAND.

This option may only be declared for User Programs. Normally when an operator enters a message containing a tranocode for a program that is not running, GEMCOS MCS displays an error message and the operator must wait until the program is executed through the Console or a Control Station. However, when ONDEMAND is selected, GEMCOS MCS ZIP-executes the program if it is not running when a tranocode message is received for it. The first message received for the program causes the execute.

ONDEMAND functions are internal and not visible to the operator. This feature enables the operator to enter messages without interruption. The messages are stored in a "tank file". When GEMCOS MCS receives a FILE OPEN for the program, all "tanked" messages for that program are sent to it in the same order as originally received by the GEMCOS MCS. The tank file is closed when it contains no more messages.

BOJ.

The BOJ (Beginning-Of-Job) option may be declared for assignment or user programs. When the GEMCDS MCS is executed, it automatically executes all BOJ programs.

NOTE

It is advisable to be selective when declaring programs BOJ so the mix is not filled with unnecessary jobs.

MANUAL.

MANUAL may be declared for Utility, Assignment and User Programs. If a program declared MANUAL is not running, it must be executed with the EX command. Utility Programs can only be declared as MANUAL.

Examples:

EXECUTE = MANUAL, BOJ.
EXECUTE = ONDEMAND.

RECOVERY STATEMENT.

Syntax:

<RECOVERY statement> ::= RECOVERY = <recovery list>. / <empty>

**<recovery list> ::= SYNCHRONIZED / DATABASE / QUEUERESTORATION /
NONE**

Semantics:

The <RECOVERY statement> declares what type of recovery mechanism (if any) this program undergoes after a system or program failure. Synchronized and data base recovery are for programs that are part of a data base. Queue-restoration recovery is for programs that are not logically associated with any other programs. The default for this statement is NONE. See section 9 for a detailed explanation of the recovery mechanism.

DATA BASE NAME STATEMENT.

Syntax:

```

<DATABASE NAME statement> ::= DATABASENAME = <database name list>.
                               / <empty>

<database name list>      ::= <database identifier> /
                               <database identifier> ,
                               <database name list>

```

Semantics:

The <DATABASE NAME statement> associates a program with a data base. If recovery for a program is synchronized or data base, this statement must be present and specify the name of the data base that the program belongs to; otherwise, it is not required. If this statement is required but not given, a syntax error occurs.

If the program is a Restart Program (RESTART PROGRAM = TRUE), then more than one data base identifier may be specified providing that the Restart Program services more than one data base. If the program is not a Restart Program, only one data base identifier may be specified. A data base identifier is an identifier that contains between 1 and 17 characters.

Examples:

```

DATABASENAME = MCSTESTDB.
DATABASENAME = LIVEDB, TESTDB.

```

AUDIT TRANSACTIONS STATEMENT.

Syntax:

```
<AUDIT TRANSACTIONS statement>
    ::= AUDITTRANSACTIONS = <trancode list>. /
    <empty>

<trancode list>
    ::= <trancode> / <trancode> ,
    <trancode list> / ALL
```

Semantics:

The <AUDIT TRANSACTIONS statement> specifies which previously defined trancodes are to be audited by the MCS. Only transactions that cause the data base to be updated should be audited, since all audited messages are reprocessed during recovery. If ALL is selected, no individual trancodes may be specified and all trancodes for this program are audited. If recovery is required for this program, then either this statement or the <AUDIT ASSIGNMENT statement> or both must be specified. By default, the MCS does not audit by trancode for any program.

Examples:

```
AUDITTRANSACTIONS = UPD.
AUDITTRANSACTIONS = PAY, DE01, DE02, DE04.
AUDITTRANSACTIONS = ALL.
```

AUDIT ASSIGNMENT STATEMENT.

Syntax:

**<AUDIT ASSIGNMENT statement> ::= AUDITASSIGNMENT = <logical value>.
/ <empty>**

Semantics:

The <AUDIT ASSIGNMENT statement> directs the MCS to audit or not to audit messages that do not have a tranocode. Programs declared as User Programs may not use this statement since all messages for that class of program necessarily contain a tranocode. User Programs that require recovery must use the <AUDIT TRANSACTIONS statement>. Programs of any other class that require recovery must use this statement or the <AUDIT TRANSACTIONS statement> or both. By default, the MCS does not audit by assignment.

Examples:

**AUDITASSIGNMENT = TRUE.
AUDITASSIGNMENT = FALSE. % FOR DOCUMENTATION ONLY**

**PROGRAM Section
cont**

AUDIT OUTPUT STATEMENT.

Syntax:

**<AUDIT OUTPUT statement> ::= AUDITOUTPUT = <logical value>. /
<empty>**

Semantics:

The <AUDIT OUTPUT statement> directs the MCS to audit all output messages from the program to the station. This statement must be set to TRUE for programs that use synchronized recovery; otherwise, a warning is issued and the statement is automatically set to TRUE. For the MCS to audit output, a program must audit either by assignment or by transaction. Except for synchronized recovery, AUDITOUTPUT defaults to FALSE.

Examples:

**AUDITOUTPUT = TRUE.
AUDITOUTPUT = FALSE.**

RESTART PROGRAM STATEMENT.

Syntax:

**<RESTART PROGRAM statement> ::= RESTARTPROGRAM = <logical value>.
/ <empty>**

Semantics:

The <RESTART PROGRAM statement> specifies whether or not this program is to be a Restart Program (see section 9 for a detailed explanation of a Restart Program and its functions). If this statement is set to TRUE, then the following must also be done:

- a. Recovery must be set to either synchronized or data base.
- b. A <DATABASE NAME statement> must be supplied. More than one data base name is allowed if this Restart Program services more than one data base, and each data base uses the same type of recovery (synchronized or data base).

Each data base must have exactly one Restart Program declared, but a Restart Program can service multiple data bases. By default, RESTARTPROGRAM is set to FALSE.

Examples:

RESTARTPROGRAM = TRUE.
RESTARTPROGRAM = FALSE.

**PROGRAM Section
cont**

MAXIMUM COPIES STATEMENT.

Syntax:

<MAXCOPIES statement> ::= MAXCOPIES = <integer>. / <empty>

Semantics:

The <MAXCOPIES statement> is used to specify the number of copies of this program that can be running (have a remote file open) at any one time. For Assignment Programs, the only allowable value is one. The sum of MAXCOPIES for all programs determines how many programs can be running in the MCS concurrently. An increase in the value assigned to MAXCOPIES during regeneration may consequently require generating and compiling so that the MCS has a larger value stack. The value of MAXCOPIES is safely lowered during regeneration. MAXCOPIES is set to one by default.

NOTE

MAXCOPIES replaces MAXRUNNING, a global option. MAXRUNNING should no longer be used.

Examples:

MAXCOPIES = 3.
MAXCOPIES = 2.

OPEN MESSAGE STATEMENT.

Syntax:

```
<OPEN MESSAGE statement> ::= OPENMESSAGE = <logical value>. /  
                               <empty>
```

Semantics:

When OPENMESSAGE is set TRUE, the program receives, as the first message in its remote file, information from GENCOS listing the Logical Station numbers of the stations which comprise the program remote file. The OPENMESSAGE consists of a Common-area header with an MCS-TYPE 6, LSN set to the number of stations in the program remote file, and TEXTSIZE set to LSN *3. The text is set to a list of 3-byte Logical Station Numbers. The OPENMESSAGE is not audited.

If the interface is set to MCS, the Common-area header is preceded by a B 1800/B 1700 MCS/Network Controller interface MCS to MCS DATA MESSAGE header with the MESSAGE TYPE FIELD set to 80. A program with an interface of NONPARTICIPATION cannot request the OPENMESSAGE. By default, OPENMESSAGE is FALSE.

Example:

```
OPENMESSAGE = TRUE.
```

**PROGRAM Section
cont**

ATTACH MESSAGE STATEMENT.

Syntax:

**<ATTACH MESSAGE statement> ::= ATTACHMESSAGE = <logical value>. /
<empty>**

Semantics:

When ATTACHMESSAGE is set TRUE, the program receives a message in its remote file giving the Logical Station number of a station which just attached itself to the program (by means of the *EX Network Control Command). The first station to attach itself does not generate an ATTACHMESSAGE. The station can be obtained from the OPENMESSAGE. The ATTACHMESSAGE consists of a Common-area header with an MCSTYPE 2, an LSN set to the Logical Station number of the attaching station, SEQNO set to the next audit sequence number, and TEXTSIZE set to 0000. No message text is sent.

If INTERFACE is set to MCS, the Common-area header is preceded by a B 1800/B 1700 MCS Network Controller interface MCS DATA MESSAGE header with the Message Type field set to 80. A program with an interface of NONPARTICIPATION cannot request ATTACHMESSAGES. By default, ATTACHMESSAGE is FALSE.

Example:

ATTACHMESSAGE = TRUE.

DETACH MESSAGE STATEMENT.

Syntax:

```
<DETACH MESSAGE statement> ::= DETACHMESSAGE = <logical value>. /
<empty>
```

Semantics:

When DETACHMESSAGE is set TRUE, the program receives a message in its remote file giving the Logical Station number of the station which has just detached itself from the program (by means of the HAP Network Control Command). The last station to detach itself does not generate a DETACHMESSAGE since the program is informed of the fact (it receives an End-of-File condition on its remote file). The DETACHMESSAGE consists of a Common-area header with an MCSTYPE of 4, an LSN set to the Logical Station number of the detaching station, SEQNO set to the next audit sequence number, and TEXTSIZE set to 0000. No message text is set.

If INTERFACE is set to MCS, the Common-area header is preceded by a B 1800/B 1700 MCS/Network Controller interface, MCS DATA MESSAGE HEADER with the Message Type field set to 80. A program with an interface of NONPARTICIPATION cannot request DETACHMESSAGES. By default, DETACHMESSAGE is FALSE.

Example:

```
DETACHMESSAGE = TRUE.
```

**PROGRAM Section
cont**

CONVERSATIONSIZE STATEMENT.

Syntax:

**<CONVERSATIONSIZE statement> ::= CONVERSATIONSIZE = <integer>.
/ <empty>**

Semantics:

The <CONVERSATIONSIZE statement> is used to establish the size of the conversation area for a program. The size is specified in bytes. The MCS cannot generate conversational capabilities without this statement in the TCL. Anytime this statement is increased to a value greater than any previously declared CONVERSATIONSIZE, the TCL must be regenerated and recompiled.

Examples:

**CONVERSATIONSIZE = 30.
CONVERSATIONSIZE = 45.**

MAXASSIGNERS STATEMENT.

Syntax:

**<MAXASSIGNERS statement> ::= MAXASSIGNERS = <integer>.
/ <empty>.**

Semantics:

The <MAXASSIGNERS statement> is used for utility programs only. This statement is used to specify the maximum number of stations that can be attached to a program concurrently. The maximum value cannot be greater than the number of stations allowed by GENCOS. By default, all attachments are applied to one program.

Examples:

**MAXASSIGNERS = 5.
MAXASSIGNERS = 2.**

**PROGRAM Section
cont**

TRANSACTION CODE POSITION STATEMENT.

Syntax:

**<TRANCODE POSITION statement> ::= TRANCODE POSITION = <integer>. /
<empty>**

Semantics:

The <TRANCODE POSITION statement> allows the user to specify where trancodes are found in messages from this program. The position specified in <integer> represents the number of characters after the common area header.

Examples:

**TRANCODEPOSITION = 6.
TRANCODEPOSITION = 1.**

AP300STATUS STATEMENT.

Syntax:

**<AP300STATUS Statement> ::= AP300STATUS = <logical value>.
/ <empty>**

Semantics:

The <AP300STATUS statement> indicates whether the four-byte AP300STATUS message from the AP300 is forwarded to the attached application program. The status of the AP300 is reported to the control station or the system SPD when the four-byte status is received. The default value is false.
false.

Example:

AP300STATUS = True.

STATION Section

STATION_SECTION.

Syntax:

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

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

<station define> ::= STATION <station name>:
                    <station description>

<station name> ::= <NDL station identifier>

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

<STATION statement list> ::= <STATION statement> /
                              <STATION statement list>
                              <STATION statement> /
                              <empty>

<STATION statement> ::= <SIGNON statement> /
                        <SCREEN SIZE statement> /
                        <TRANCODE POSITION statement> /
                        <VALID ACCESS KEYS statement> /
                        <TRANSACTION MODE statement> /
                        <CONTINUOUS LOG ON statement> /
                        <CONVERSATIONAL statement> /
                        <TRANCODE statement> /
                        <TYPE statement>
```

Semantics:

The <STATION section> must be present to define various attributes of stations which the MCS is to service. (A GEMCDS MCS opens a remote file whose name is given in the <QUEUE NAME statement>.) In the FAMILY statement of the FILE SECTION of the user's NDL source, this remote file was assigned a station identifier list. These are the stations which the MCS services and which must be defined in the Transaction Control Language <STATION section>.

The <STATION section> is composed of a <station define list>. Each <station define> describes one station. The <station name> must be the station identifier used to refer to that station in NDL. The <STATION statement list> is optional. One of the <station names> may appear in the <CONTROL STATIONS statement> of the <GLOBAL section>.

Example_:

```
STATION TD800A:  
  SIGNON = TRUE.  
  SCREENSIZE = 1024.  
  TRANCODEPOSITION = 5.  
  VALIDACCESSKEYS = ABCD, XXYY, 84080.  
STATION TD800B:  
  SCREENSIZE = 1920.
```

**STATION Section
cont**

SIGN-ON STATEMENT.

Syntax:

<SIGNON statement> ::= SIGNON = <logical value>. / <empty>

Semantics:

The <SIGNON statement> indicates whether a user must sign on at this station prior to entering messages. If SIGNON is TRUE, the operator must sign on with one of the <access codes> listed in the <VALID ACCESS KEYS statement>. If VALIDACCESSKEYS is set to ALL, the operator must sign-on with one of the <access codes> listed in the <ACCESS CONTROL statement>. The <SIGNON statement> is optional and, if omitted, defaults to FALSE.

Examples:

**SIGNON = TRUE.
SIGNON = FALSE.**

SCREEN SIZE STATEMENT.

Syntax:

<SCREEN SIZE statement> ::= SCREENSIZE = <integer>. / <empty>

Semantics:

The <SCREEN SIZE statement> defines the length of the largest message which may be received by this station. If the MCS determines that a message larger than <integer> characters is bound for the station, the message is broken into several transmissions until the entire message is sent.

CAUTION

If any <station define> has a <SCREEN SIZE statement> in its (STATION statement list), all <station defines> must have one. If the <SCREEN SIZE statement> is not present, no screen wraparound occurs.

The occurrence of a <SCREEN SIZE statement> causes the screen-wraparound code to be generated into the MCS code file. If no station is defined as having a SCREENSIZE less than or equal to MAXTEXT SIZE, the <SCREENSIZE statement> should be avoided. The result is a more efficient MCS.

Examples:

SCREENSIZE = 1920.
SCREENSIZE = 256.

STATION Section
cont

TRANSACTION CODE POSITION STATEMENT.

Syntax:

<TRANCODE POSITION statement> ::= TRANCODEPOSITION =
<integer>. / <empty>

Semantics:

The <TRANCODE POSITION statement> allows the user to specify where
trancodes are to be found in messages received from this station.
By default, TRANCODEPOSITION is 1.

Examples:

TRANCODEPOSITION = 5.
TRANCODEPOSITION = 1.

VALID ACCESS KEYS STATEMENT.

Syntax:

```

<VALID ACCESS KEYS statement>
    ::= VALIDACCESSKEYS = <access key list>. /
    <empty>

<access key list>
    ::= <access code> / <access code> ,
    <access key list> / ALL

<access code>
    ::= <identifier>
  
```

Semantics:

A list of valid access keys can be prepared to enforce access key validation at sign-on time. Each <access code> which appears in a <VALID ACCESS KEYS statement> must have occurred in the <ACCESS CONTROL statement>. ALL indicates that any <access code> can be used to sign on at this station. If the statement is omitted and SIGNON is TRUE, ALL is assumed. This statement has no meaning if SIGNON is FALSE.

Examples:

```

VALIDACCESSKEYS = ALL.
VALIDACCESSKEYS = 84080, 84090, ABCD.
  
```


TRANSACTION MODE STATEMENT.

Syntax:

**<TRANSACTION MODE statement> ::= TRANSACTIONMODE = <logical value>.
/ <empty>**

Semantics:

The <TRANSACTION MODE statement> determines whether or not a station is allowed to transmit a new input transaction 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 the receipt and transmission by the MCS of the response to the current transaction for the station. Also, this station can only send messages to programs that are declared to use synchronized recovery. TRANSACTIONMODE being TRUE is ignored if auditing is not present in the MCS. By default, TRANSACTIONMODE is set to FALSE.

Examples:

**TRANSACTIONMODE = TRUE.
TRANSACTIONMODE = FALSE.**

CONTINUOUS LOG-ON STATEMENT.

Syntax:

```
<CONTINUOUS LOG ON statement> ::= CONTINUOUSLOGON =  
                                <logical value>. / <empty>
```

Semantics:

The <CONTINUOUS LOG ON statement> is used to determine whether the MCS should "remember" who was logged on to the station following a termination of the network (either normally or abnormally). After the network is restarted, if CONTINUOUSLOGON is TRUE for a station, the user remains logged on. Otherwise, any users who were logged on at the time of failure will be logged off. CONTINUOUSLOGON being set to TRUE is ignored if auditing is not present in the MCS. By default, CONTINUOUSLOGON is set to FALSE.

Examples:

```
CONTINUOUSLOGON = TRUE.  
CONTINUOUSLOGON = FALSE.
```

CONVERSATIONAL STATEMENT.

Syntax:

**<CONVERSATIONAL statement> ::= CONVERSATIONAL = <logical value>.
/ <empty>**

Semantics:

The <CONVERSATIONAL statement> determines whether a station can participate in a conversation. By default, the statement is set to TRUE.

Examples:

**CONVERSATIONAL = TRUE.
CONVERSATIONAL = FALSE.**

TRANCODE STATEMENT.

Syntax:

```
<TRANCODE statement> ::= TRANCODE = <trancode list>. / <empty>  
<trancode list> ::= <trancode> / <trancode>,<trancode list>
```

Semantics:

The <TRANCODE statement> is used to define trancodes and to associate them with stations. A trancode identifier is any string up to ten characters in length. A trancode defined in a <TRANCODE statement> may occur in the <ACCESS CONTROL statement> to restrict its use to a specific list of <access keys>.

By using these trancodes, messages from another station or program can be routed to this station.

NOTE

The <module-function indicies> are not applied to trancodes.

Examples:

```
TRANCODE = STATION, STATION2, HELLO.  
TRANCODE = HELP.
```

STATION Section
cont

TYPE STATEMENT.

Syntax:

**<TYPE statement> ::= TYPE = <type list>. / <empty>
<type list> ::= AP300 / MT600 / STANDARD**

Semantics:

The <TYPE statement> is used to define the physical type of each station. AP300 and MT600 are standard Burroughs terminal devices. (Refer to section 10 for further information about this statement.)

Examples:

TYPE = AP300.
TYPE = ROUTEHEADER.

DEVICE_SECTION.

Syntax:

```

<DEVICE section> ::= <device define list>

<device define list> ::= <device define> /
                        <device define list> <device define>

<device define> ::= device <device name> :
                  <device description>

<device name> ::= <identifier>

<device description> ::= <DEVICE statement list>

<DEVICE statement list> ::= <DEVICE statement> /
                            <DEVICE statement list>
                            <DEVICE statement>

<DEVICE statement> ::= <STATION list statement> /
                      <INPUT FORMATS statement> /
                      <OUTPUT FORMATS statement>

```

Semantics:

The <DEVICE section> is used to group stations by device class and to indicate which format is to be applied to a message. The <DEVICE section> should be present if and only if the <FORMAT AND FUNCTION statement list> is present in the <GLOBAL section>. The <DEVICE section> may never occur if the 8 1800/B 17)0 GEMCDS release being used is not an advanced version.

Each <device define> consists of a <STATION LIST statement>, an <INPUT FORMATS statement> and an <OUTPUT FORMATS statement>. The <device name> may be any identifier and is not referenced elsewhere in TCL.

In the following example, an input message from TD800A or TD800B with tranocode INQ or UPDATE is formatted using format X. A message from TD700A, TD700B or TD700C with tranocode INQ or UPDATE has format Y applied.

Similarly on output, if a program writes a message with the message-ID field of the Common-area header set to "PAY" or "SHIP" (left justified with trailing blanks), format X12 is used when the message is bound for TD800A or TD800B. Format Y12 is applied if the message is bound for a station in the device class TD700 (TD700A, TD700B or TD700C). When the message-ID field of the Common-area header is set to "RCV", X33 or Y33 is applied, again depending on the destination of the message.

Finally, when an operator transmits "PAY" or "SHIP" (without leading or trailing blanks) from station TD800A or station TD800B, the operator is making a forms request. Using format X12, a message is built up with all blank data fields (A, B, I, J and T). The message is sent to the requesting station. Format Y12 is used to create the blank-formatted screen if the forms request for "PAY" or "SHIP" came from TD700A, TD700B, or TD700C. If a "RCV" forms request is received, format X33 or Y33 is applied, depending on the device classification of the requesting station.

NOTE

Formats X, X12, X33, Y, Y12 and Y33 must have been defined in the <FORMAT AND FUNCTION statement list>.

Example:

```
DEVICE TD800:
  STALIST = TD800A, TD800B.
  FORMATSIN:
    X = INQ, UPDATE.
  FORMATSOUT:
    X12 = PAY, SHIP.
    X33 = RCV.
DEVICE TD700:
  STALIST = TD700A, TD700B, TD700C.
  FORMATSIN:
    Y = INQ, UPDATE.
  FORMATSOUT:
    Y12 = PAY, SHIP.
    Y33 = RCV.
```

STATION LIST STATEMENT.

Syntax:

<STATION LIST statement> ::= STALIST = <station name list>.
<station name list> ::= <station name> /
<station name list>, <station name>
<station name> ::= <identifier>

Semantics:

The <STATION LIST statement> specifies the stations which are to be considered part of each device class. Each <station name> must be defined in the <STATION section>, and each <station name> occurring in the <STATION section> should appear in exactly one <STATION LIST statement>.

Example:

STALIST = TD820A, TD820B, TD820C.

INPUT FORMATS STATEMENT.

Syntax:

```
<INPUT FORMATS statement> ::= FORMATSIN: <input format list>.  
  
<input format list> ::= <input format association> /  
                        <input format list>  
                        <input format association>  
  
<input format association> ::= <format-ID> =  
                                <trancode list>  
  
<format-ID> ::= <identifier>  
  
<trancode list> ::= <trancode>  
                    <trancode list>,  
                    <trancode list>  
  
<trancode> ::= <trancode identifier>
```

Semantics:

The <FORMATSIN statement> indicates which format is to be applied to a particular message entered at a station of a particular device class before the message is forwarded to the appropriate program. Only messages with a recognizable trancode are formatted. Each <format-ID> must be defined in the <FORMAT AND FUNCTION statement list>, and each trancode must be defined in the <PROGRAM section>. A trancode may only be associated with one <format-ID> per <FORMATSIN statement>.

The MCS determines if an input message is to be formatted by attempting to recognize a trancode in the message text. When a trancode is found, the message is formatted only if the trancode was associated with a <format-ID> in the device class determined by the station where the message originated.

Example:

```
FORMATSIN:  
  MT1 = PAY1, PAY2.  
  FMT2 = INV1.  
  FMT3 = INV2, INV3.
```

OUTPUT FORMATS STATEMENT.

Syntax:

```

<OUTPUT FORMATS statement> ::= FORMATSOUT: <output format list>.
<output format list>      ::= <output format association> /
                             <output format list>
                             <output format association>
<output format association> ::= <format-ID> = <msg-ID list>
<format-ID>                ::= <identifier>
<MSG-ID list>              ::= <message-ID> /
                             <msg-ID list> , <message-ID>
<message-ID>               ::= <identifier>

```

Semantics:

The <FORMATSOUT statement> indicates which format is to be applied to a particular message written by a program bound for a station of a particular device class. Only messages with a recognizable <message-ID> in the message-ID field of the Common-area header are formatted. Each <format-ID> must be defined in the <FORMAT AND FUNCTION statement list>. A <message-ID> may only be associated with one <format-ID> per <FORMATSOUT statement> and cannot exceed six characters in length. Station operators can enter a message consisting solely of a <message-ID> and, in doing so, make a forms request.

When the MCS receives a program message, the message-ID field in the header is checked. If the field is filled, the contents cause the MCS to format the message according to the <format-ID> associated with the <message-ID> in the device class. The device class is determined by the station to which the message is sent.

When the MCS receives a message one to six characters in length without a recognizable trancode from a station, a check is made to determine if the message consists of a <message-ID>. If so, the operator entered a forms request. The MCS builds a message with blank data fields using the <format-ID> with which the <message-ID> was associated in the device class determined by the station from which the forms request was received.

Example:

```

FORMATSOUT:
  FOR1 = PAY8, PAY9.
  FMT1 = INV1.

```

MESS_CODE_SECTION.

Syntax:

```
<MESS CODE section> ::= <static declarations>
                        <dynamic declarations>
                        <procedure define list>
```

Semantics:

The <MESS CODE section> enables the user to include UPL-mergeable external source statements to supplement or replace GEMCOS MCS functions. Mergeable external source statements are for specialized requirements which demand deviation from the standard GEMCOS logic. User-written MESS procedures can be merged into key locations in the MCS source code file.

NOTE

Changes made to the <MESS CODE section> during a REGENERATE MCSTCL run do not affect the MCS source code file.

The <MESS CODE section> consists of <static declarations>, <dynamic declarations> and a <procedure define list>. The <MESS CODE section> is optional.

Example:

```
STATIC DECLARATIONS:
  DECLARE
    X                FIXED;
ENDSOURCECODE.
DYNAMIC DECLARATIONS:
  DECLARE DYNAMIC
    SPEC.STRING      CHARACTER(x);
ENDSOURCECODE.
PROCEDURE SETSIZES:
  PROCEDURE MESS.SET.SIZES;
  %
    DECLARE
      ACCEPT.STRING   CHARACTER(5);
  %
    DISPLAY "ENTER STRING SIZE, XXXXX";
    ACCEPT ACCEPT.STRING;
    X := BINARY(ACCEPT.STRING);
    END MESS.SET.SIZES;
ENDSOURCECODE.
PROCEDURE SETVALUES:
  PROCEDURE MESS.SET.VALUES;
  %
    SPEC.STRING := " ";
    END MESS.SET.VALUES;
ENDSOURCECODE.
```

STATIC DECLARATIONS.

Syntax:

```

<static declarations> ::= <static declarations card>
                        <declaration source cards>
                        <end card> / <empty>

<static declarations card> ::= static declarations:

<declaration source cards> ::= <declaration source card> /
                              <declaration source cards>
                              <declaration source card>

<declaration source card> ::= <UPL DECLARATION statement> /
                              <UPL DEFINE statement> /
                              <UPL FILE statement>

<end card> ::= ENDSOURCECODE.
  
```

Semantics:

In <static declarations> the user may make UPL global declarations (except dynamic declarations), global defines and file declarations. The TCL source cards containing UPL source statements must be surrounded by a TCL <static declarations card> and a TCL <end card>. The <static declarations> are optional.

If <static declarations> are present, the <NAME STACK ENTRIES statement> and <VALUE STACK BITS statement> of the <GLOBAL section> should be set appropriately.

Example:

```

STATIC DECLARATIONS:
  DECLARE
    SECURITY.FILE.OPENED      BIT(1);
  FILE
    SECURITY (LABEL = "MCSSEC",
             DEVICE = DISK RANDOM,
             RECORDS = 80/1,
             BUFFERS = 1);
  DEFINE
    F          AS #FIXED#,
    C          AS #CHARACTER#;
ENDSOURCECODE.
  
```

MESS CODE Section
cont

DYNAMIC DECLARATIONS.

Syntax:

```
<dynamic declarations> ::= <dynamic declarations card>
                           <dynamic declaration source cards>
                           <end card> / <empty>

<dynamic declaration card> ::= dynamic declarations:

<dynamic declaration source cards>
                               ::= <dynamic declaration source card> /
                                   <dynamic declaration source cards>
                                   <dynamic declaration source card>

<dynamic declaration source card>
                                   ::= <UPL DYNAMIC DECLARATION statement>

<end card>                       ::= ENDSOURCECODE.
```

Semantics:

In <dynamic declarations> the user may make UPL dynamic declarations. The TCL source cards containing UPL source statements must be surrounded by a TCL <dynamic declarations card> and a TCL <end card>. The <dynamic declarations> are optional.

If <dynamic declarations> are present, the <NAME STACK ENTRIES statement> and <VALUE STACK BITS statement> of the <GLOBAL section> should be set appropriately.

Example:

```
DYNAMIC DECLARATIONS:
  DECLARE DYNAMIC
    WORK.AREA          CHARACTER(MS.NPR.MAX.TEXT.SIZE+250);
ENDSOURCECODE.
```

PROCEDURE DEFINE LIST.

Syntax:

```

<procedure define list> ::= <procedure define> /
                           <procedure define list>
                           <procedure define>

<procedure define> ::= <procedure introduction card>
                       <UPL PROCEDURE statement>
                       <end card> / <empty>

<procedure introduction card>
                       ::= procedure <mess procedure-ID> :

<MESS procedure-ID> ::= AUDIT / CLOSEACTION /
                       CLOSEFILES / ERRORHANDLER /
                       MSGFROMPROGRAM / MAINTENANCE /
                       MSGFROMSTATION / OPENACTION /
                       HANDLERECALL / INITIATERESTORE /
                       RESTOREPROGRAM / SETSIZES /
                       SETVALUES

<end card> ::= ENDSOURCECODE.

```

Semantics:

The <procedure define list> contains user-coded UPL procedures, one per <procedure define>. Each <procedure define> begins with a TCL <procedure introduction card>, follows with the user's source statements and ends with a TCL <end card>. The <mess procedure-ID> of the <procedure introduction card> specifies to TCL where the user's code is to be merged. The <procedure define list> is optional.

MESS CODE Section
cont

If <procedure define list> is present, the <NAME STACK ENTRIES statement> and <VALUE STACK BITS statement> of the <GLOBAL section> should be set appropriately to reflect the space required for variables declared within any of the user-written procedures.

A discussion of each of the MESS procedures is given later in this section.

Example:

```
PROCEDURE MSGFROMSTATION:
  PROCEDURE ZIP.IT BIT(1);
  %
  IF MS.MSG.TEXT.SIZE GTR 3
    THEN
      IF SUBSTR(MS.MSG.TEXT,0,3) EQL "ZIP"
        RETURN (1);
      END;
  RETURN (0);
END ZIP.IT;
ENDSOURCECODE.
```

MESS PROCEDURES.

This discussion explains when MESS procedures are called, the parameters required to call them, and the values they must return. For an explanation of how to include MESS procedures, refer to GENERAL DISCUSSION of TRANSACTION CONTROL LANGUAGE in this section. All MESS procedures are optional.

Several considerations common to each MESS procedures are now discussed.

Care must be taken when writing MESS code to avoid duplicating identifiers already used in the MCS. For this reason, it is useful to know that all MCS identifiers adhere to the following conventions:

- a. DEFINES begin with three characters: "MD." or "MS."
- b. files begin with three characters: "MCS".
- c. Data names begin with three characters: "MS."
- d. Procedure names begin with four characters: "MCS."
- e. DO-group labels begin with three characters: "ML."

MESS procedures have access not only to entities declared in <static declarations>, <dynamic declarations>, and locally, but also to many data areas, files and procedures used by the standard MCS modules according to the scope rules of UPL.

When programming MESS code, segmenting procedures must be followed. Some efficiency in memory allocation may be realized if MESS procedure segments are approximately 800 to 1200 bytes in size, the average size of standard MCS modules.

If global or local variables are declared by the user in the <MESS CODE section>, the <NAME STACK ENTRIES statement> and <VALUE STACK BITS statement> of the <GLOBAL section> should be set appropriately.

SET SIZES.

This procedure is given control during the first phase of initialization logic in the MCS. Its purpose is to specify sizes for any dynamic variables declared in the <dynamic declarations>.

Example:

Assume that <static declarations> include:

```
DECLARE    MAX.SIZE                FIXED;
```

and that <dynamic declarations> include:

```
DECLARE DYNAMIC USER.AREA        CHARACTER(MAX.SIZE);
```


MESS Procedures
cont

Then the SETSIZES routine must be provided. The <procedure define list> might include:

```
PROCEDURE SETSIZES:
  PROCEDURE MY.SETSIZES.ROUTINE;
    MAX.SIZE := 100;
  END MY.SETSIZES.ROUTINE;
ENDSOURCECODE.
```

The SETSIZES MESS procedure is called from MCS.INITIATE.SIZES.

SET_VALUES.

This procedure is given control during the second phase of initialization logic in the MCS. The purpose is to specify values for variables that were declared in the <static declarations>.

Examples:

Suppose the <static declarations> include the following:

```
DECLARE MAX.VALUES      FIXED,
        USER.DATA      CHARACTER(10);
```

Consequently, the SETVALUES routine must be provided. The <procedure define list> can include the following:

```
PROCEDURE SETVALUES:
  PROCEDURE MY.SETVALUES.ROUTINE;
    MAX.VALUES := 200;
    USER.DATA := "SOMETHING";
  END MY.SETVALUES.ROUTINE;
ENDSOURCECODE.
```

The SETVALUES MESS procedure is called from MCS.INITIALIZE.TABLES.

MESSAGE FROM STATION.

This procedure is called upon receipt of a message from a station. If MSGFROMSTATION is given control, it may assume that there is no data-communication error associated with the message, the MCS is not being shutdown, the source station is signed on if sign-on is required at that station, and the message does not begin with the signal character. The Common-area header to be associated with this message was built in MS.COMMON.AREA (but not yet attached to the message). If a trancode is present, it was recognized and noted in MS.TRN.INDEX. The message was not yet formatted and could be a forms request (advanced version only). The message was not yet audited.

MSGFROMSTATION must be a function procedure which returns a 1-bit value specifying disposition of the message:

- a. A value 0 (zero) directs the MCS to continue processing the message as if there had been no MSGFROMSTATION procedure.
- b. A value 1 signifies that the MSGFROMSTATION procedure has taken full responsibility of the message. The MCS discontinues processing this message.

If a station is associated with or was attached to the remote file of a program that uses a NONPARTICIPATION or MCS interface, the GEMCOS MCS does not receive any messages from that station. Thus, the MCS cannot pass control to MSGFROMSTATION for such messages.

This procedure can set fields in the USERAREA (MS.COMMON.USER) of the Common-area header. It can perform specialized routing, access control or input formatting, and thus be used for data collection.

MESSAGE FROM PROGRAM.

This procedure is called when the MCS receives a message from an Application Program (the MCS does not receive messages from programs using the NONPARTICIPATION or MCS interface). The message was not formatted (advanced version) nor was it audited. The message may be a request for message restoration or may contain a signal character.

NESS Procedures
cont

MSGFROMPROGRAM must be a function procedure which returns a 1-bit value specifying the disposition of the message:

- a. A value 0 (zero) informs the MCS to continue processing the message as if there had not been a MSGFROMPROGRAM procedure.
- b. A value 1 means that the message was completely processed. The MCS exits immediately to the main control procedure and does not process this message any more.

This procedure can be used to interpret the USERAREA field (MS.COMMON.USER) of the Common-area header. MSGFROMPROGRAM can perform specialized output formatting. Nonstandard routing (e.g., program-to-program message switching) can be performed using this procedure.

MAINTENANCE.

This procedure is given control when a message containing the signal character is received from a station or a program (except for valid SGN messages). The MAINTENANCE procedure is called from the standard maintenance module (MCS.MAINT.CONTROLLER) if generated. The standard maintenance module is generated if either CHANGEREQUESTS, DATADUMP, MESSAGEBROADCAST, MESSAGERECALL, PROGRAMBOJEOJ, STATUSREPORTS, SYSTEMHALT, RESTORATION, or MONITORTRACE is TRUE, or if an <ACCESS CONTROL statement> is present. When the standard maintenance module is not generated, the MAINTENANCE procedure is invoked by MCS.MSG.FROM.STATION or MCS.MSG.FROM.USER.PROGRAM.

The MAINTENANCE procedure must be a function procedure which returns a 1-bit value specifying the disposition of the message:

- a. A value 0 (zero) indicates that MAINTENANCE did not process the message. If the standard maintenance module is present, it attempts to scan for a standard command and process it. If the standard maintenance module is not present, the input is ignored.
- b. A value 1 causes the MCS to consider the message completely processed whether or not the standard maintenance module is present.

MAINTENANCE must accept one input parameter: the number of the calling procedures (FIXED).

This procedure can be used to implement new Network Control Commands or change existing commands.

AUDIT.

This procedure can either supplement or replace the standard auditing logic. It is called after the standard audit procedure (if generated) is executed. Any files needed by AUDIT must be declared in the <GLOBAL section> of MESS code.

AUDIT must accept two parameters:

- a. The number (FIXED) of the calling procedure.
- b. An indicator [CHARACTER (2)] which denotes the type of message being sent from the MCS:
 - 1) A value of 00 indicates the message is bound for a station.
 - 2) A value of 01 indicates the message is bound for a program.
 - 3) Other values denote special communications between the MCS and the Network Controller.

ERROR HANDLER.

The primary function of this procedure is to supplement the standard error handling module of the MCS. As a secondary function it may also save the contents of MS.MSG.WORK.AREA in case of a data dump. For certain system errors, a data dump is created. The standard error handling logic induces the dump by synthesizing an RDM Network Control Command in MS.MSG.WORK.AREA. The message that was stored in MS.MSG.WORK.AREA when the error was detected is therefore lost, unless the ERRORHANDLER procedure saves it.

This routine must accept one parameter: the error message number [CHARACTER(2)] which corresponds to the error detected.

ERRORHANDLER must be a function procedure that returns a value [BIT (1)] which tells the MCS what to do with the error condition:

- a. A value 0 (zero) indicates that the error should be processed normally.
- b. A value of 1 indicates that the MCS is to exit the error handling module immediately.

The ERRORHANDLER procedure is called by MCS.PRINT.ERROR.

CLOSE FILES.

The CLOSEFILES procedure is given control during system shutdown (EOJ). Its primary purpose is to close any files that may have been opened in other MESS routines. This is called only by the MCS.EOJ procedure. It is not called unless the generation parameter SYS-HALT is set.

HANDLE RECALL.

The HANDLERECALL procedure is given control during system shutdown (EOJ). At this time, the MCS recalls all messages which have been routed to a station but are still awaiting transmission in the queues of the Network Controller. The HANDLERECALL procedure is invoked each time a message returns to the MCS.

The HANDLERECALL procedure must be a function procedure which returns a value [BIT(1)] specifying disposition of the message:

- a. A value 0 (zero) causes the MCS to print the message on a line printer before discarding it.
- b. A value 1 causes the MCS to discard the message without further processing.

If this procedure is not provided, the MCS prints all recalled messages before discarding them.

This procedure is called only by MCS.EOJ. It is not called unless the generation parameter SYS-HALT is set.

INITIATE RESTORE.

This procedure is given control when an application program indicates to the MCS that it needs restoration (by sending a message which has the MCSTYPE field of the Common-area header set to 1). It may either supplement or replace the module that performs restoration initialization, depending on the <RESTORATION statement>. Its purpose is to perform any initialization that may be necessary to prepare for restoration.

This procedure is called only by MCS.MSG.FROM.USER.PROGRAM.

RESTORE PROGRAM.

This procedure is intended to replace the standard MCS restoration logic. It is called from the main processing loop in MCS.MODULE.MANAGER. It is called once in each iteration of the loop as long as the flag MS.RESTORE.PROGRAM has a value of 1. If it is necessary to handle other network activity during restoration, MESS.RESTORE.PROGRAM must relinquish control (i.e., RETURN) occasionally so that the main processing loop can run through another cycle.

OPEN ACTION.

This procedure is intended to replace or supplement the action taken by the MCS after a FILE OPEN STATION ATTACH is approved. Normally, the MCS sends a "good day" message to each station in the newly opened file and, if specified in TCL, a FILE OPEN notification is sent to the program.

The OPENACTION procedure must be a function procedure which returns a value [BIT (1)].

- a. A value 0 (zero) causes the MCS to send the "good day" messages and the open notification.
- b. A value 1 causes the MCS to skip the code which sends the "good day" messages and the open notification.

If the action taken by the OPENACTION procedure depends on whether it is called through a FILE OPEN or through a STATION DETACH, the source of the call can be determined by checking the data field MS.MSG.HDR.TYPE. If the field contains 10, the OPENACTION procedure has been called through a FILE OPEN. If it contains any other value, it has been called through a STATION ATTACH.

CLOSE ACTION.

This procedure is intended to supplement the MCS remote file close or STATIONDETACH logic. It is called after the MCS performs the necessary steps to verify that a program is no longer on-line. The CLOSEACTION procedure must be a function procedure which returns a value [BIT (1)]. However, the value of the return is of no consequence and is reserved for future use. One useful function of the CLOSEACTION procedure might be to construct and send a notification of the FILE CLOSE to the stations involved.

If the action taken by the CLOSEACTION procedure depends on whether it is called through a remote FILE CLOSE or through a STATIONDETACH, the source of the call can be determined by checking the data field MS.MSG.HDR.TYPE. If this field contains 16, the CLOSEACTION procedure has been called through a FILE CLOSE. If it contains any other value, it has been called through a STATIONDETACH.

Common-area Header

COMMON-AREA HEADER.

The Common-area header precedes all messages sent to programs using the PARTICIPATION interface, and it is required in front of all messages written by such programs. The length of the Common-area header can vary from 60 to 200 bytes by program as specified in the <COMMONSIZE statement>. The layout of the Common-area header is as follows:

```
01  COMMONAREA.
    05  MSGDESTINATION      PICTURE 9(1).
    05  LSN                 PIC 9(3).
    05  PGMNBR REDEFINES LSN PIC 9(3).
    05  MTSMSGTYPE         PIC S9(1).
    05  SEQNO              PIC 9(6).
    05  NDLTIME            PIC 9(7).
    05  TEXTSIZE           PIC 9(4).
    05  TERMTYPE           PIC 9(2).
    05  MSGID              PIC X(6).
    05  INDEX1             PIC 9(2).
    05  INDEX2             PIC 9(2).
    05  ERROR              PIC 9(1).
    05  FMterr             PIC 9(1).
    05  MCSTYPE            PIC 9(2).
    05  INPUTADDR          PIC 9(9).
    05  RETRYCOUNT        PIC 9(1).
    05  RECOVERYSTATUS     PIC 9(1).
    05  OUTPUTADDR         PIC 9(9).
    05  CONVERSATIONSTATUS PIC 9(1).
    05  CONVERSATIONBOJEQJ PIC 9(1).
    05  USERAREA          PIC X( ).
```

The following is a detailed explanation of each field in the Common-area header:

- a. **MSGDESTINATION** - This field can be filled in by the application program to indicate special routing. It is used primarily for program-to-program or program-to-station tranocode routing. The GEMCDS system fills the field with the default value before sending it to the program. Thus, the application program need not adjust the value unless special routing is required. A list of the values for this field and the default values, set by GEMCDS, follow. (For descriptions of these fields, refer to section 4.)
 - 0 - Send to indicated station (final destination - no default)
 - 1 - Send to indicated program (final destination - no default)
 - 2 - Route with tranocode (final destination - no default)
 - 3 - Route with tranocode. Return to station (the GEMCDS system sets the value to zero).

- 4 - Route with trancode, return to program (GEMCDS sets default to 0)
 - 5 - Program can not send this value. It is set by GEMCDS to indicate that the message originates from a route-header station. It should not be altered unless an intermediate transaction is required. See section 10 for explanation of routeheader stations.
- b. LSN (logical Station Number). For incoming messages or recovered incoming messages, this field contains the LSN of the originating station. Outgoing messages are sent to the station whose LSN is stored in this field. For attach notifications and detach notifications, this field contains the LSN of the station involved. For open notifications, this field contains the number of stations in the approved FILE OPEN.
- c. PGMNBR (program number) - This field contains the program number of the originating program in the event that the message must be routed back to the program. It redefines the LSN field, so that no LSN is present if a program number is specified. MSGDESTINATION will be 1.
- d. MTSMSGTYPE (modular terminal system message type) - This field is used to identify incoming and outgoing messages when the source or destination is an MTS terminal. Refer to section 10 (station types) for a detailed explanation of this field.
- e. SEQNO (Sequence Number). The MCS assigns a unique number to each message. That number is passed to the application program in this field.
- f. NDLTIME. This is the time that the Network Controller sends the message to the MCS.
- g. TEXTSIZE. For incoming messages, this is the length in characters of the message text. It does not include the length of the Common-area header. For open notification, it is set to the LSN field multiplied by 3. When the application program writes a message, it must use the ACTUAL KEY of the remote file to specify the text size. In this case, the size must include the size of the Common-area header.
- h. TERMTYPE (Terminal Type). The MCS sets this field on incoming messages to a code which identifies the type of the originating device. Terminal-type codes are assigned in the Terminal section of NDL.
- i. MSGID (Message-ID). The MCS sets this field to blanks for incoming messages. If the program sets this field to a valid <message-ID> (refer to the <OUTPUT FORMATS statement>), the MCS formats the message before transmitting it to a station.

**Common-area Header
cont**

If the program leaves the field blank, the MCS does not format the message.

- j. INDEX1 (Module-Function Index One). If an incoming message contains a valid trancode and that trancode has module function indicies defined in TCL, the MCS sets this field to the first index; otherwise, this field is set to zero.
- k. INDEX2 (Module-Function Index Two). If an incoming message has module function indicies defined in TCL, the MCS sets this field to the second index; otherwise, this field is set to zero.
- l. ERROR - If the MCS detects an error while routing (by trancode) a message from a program, a value is returned. Definitions follow for these values:
 - 0 - No error
 - 1 - Missing trancode (trancode routing was specified)
 - 2 - Requested program or station not available
 - 3 - Return station ID is invalid
 - 4 - Error in routeheader (processor to processor) message
- m. FMterr (Format Error Indicator). If the MCS detects an error while formatting an incoming message, this field is set. Note that errors detected while formatting an outgoing message are reported to the Control station. Refer to the discussion of formatting errors under <format declaration> for an explanation of the values which can be found in this field.
- n. MCSTYPE (Message Type). The message type field identifies the type of message being exchanged between the user application program and the MCS. The allowed values and their meanings are:

<u>Value</u>	<u>Meaning</u>
0	On input, this is a message from a station. On output, this is the last (primary) message for the current transaction.
1	Not used on input. On output, this is a secondary message (i.e., the program has additional responses to send for this transaction).
2	On input, this is a station attach notification. Not used on output
4	On input, this is a station detach notification. Not used on output.
6	On input, this is a file open notification. Not used on output
15	On input, this message instructs the Restart Program to pass recovery information back to the MCS. Not used on output
17	Not used on input. On output, this message is sent by the Restart Program to the MCS. It contains recovery information requested by the MCS.
18	Not used on input. On output, this message is sent by the Restart Program to inform the MCS that an error was found.
20	Not used on input. On output, this message is sent to the MCS to indicate that the user application program needs recovery.
21	On input, this message instructs the user application program to prepare for recovery. Not used on output
22	Not used on input. On output, this message is sent by the user application program to inform the MCS that it is ready for recovery (used in response to a type 21 message only).

<u>Value</u>	<u>Meaning</u>
23	On input, this message is sent by the MCS to the user application program immediately after the remote file is opened. Its purpose is to pass information to the program that must be saved in the restart data set. Not used on output
24	On input, this message is sent to the user application program instructing it to close its data base and prepare to terminate processing. Not used on output
25	Not used on input. On output, this message is sent by the user application program to inform the MCS that the program has successfully closed its data base and is now ready to terminate processing.

- o. INPUTADDR (Input Audit Disk Address). This field contains the audit-file disk address of this transaction. If this field is zero, this transaction was not audited.
- p. RETRYCOUNT (Transaction Retry Count). This field contains the number of times this transaction was submitted to the user application program. The value is incremented by one whenever an input transaction causes a user application program to abort.
- q. RECOVERYSTATUS (System Recovery Status). This field indicates the system status at the time this transaction was sent. The allowed values and meanings are:

<u>Value</u>	<u>Meaning</u>
0	The system is not in recovery mode.
1	The system is in recovery mode caused by a user application program abort.
2	The system is performing an archival recovery.
3	The system is in recovery mode caused by a Clear/Start or an abnormal termination of the MCS.

<u>Value</u>	<u>Meaning</u>
--------------	----------------

- | | |
|----|--|
| r. | OUTPUTADDR (Output Audit Disk Address). This field contains the audit file disk address of the output message generated by the user application program. This field is not used by the program. |
| s. | CONVERSATIONSTATUS. This field indicates whether the conversation path is clear, a conversation is in progress, or an error occurred by the last message. Descriptions follow for each value that is possible in this field: <ul style="list-style-type: none">0 - Path is clear. There is no conversation in progress at the station, or the station is nonconversational.1 - Conversation in progress. The value of the CONVERSATIONBOJEOJ field indicates whether the station is communicating with a program.2 - Error. Maximum number of conversations was exceeded. The last message is neither audited nor delivered, but returned.3 - Error. Conversation is attempted with a nonconversational station. Message is returned.4 - Error. Conversation attempted with a conversing station. Message is returned.5 - Error. A nonconversational program attempts to initiate a conversation. Message is returned to the program. |

Common-area Header
cont

t. CONVERSATIONBOJE0J. This field indicates the beginning and the end of a conversation. The field is kept up-to-date through the messages from the user. Descriptions follow for each value that is possible in the field:

1) For messages to stations:

0 - End of conversation, or no conversations in progress. The MCS expects message text immediately after the common area header.

1 - Conversation is beginning or continuing. Conversation text is located between the common area header and the message text. Conversation text is stored in the conversation area. If GEMCDS is auditing the participating program, the conversation text is audited as well.

2) For messages from stations:

0 - Unoccupied. By returning this value, the station indicates it is open for conversation.

1 - Occupied. This value verifies to the program that it is in conversation with the station. Conversation text follows the common area header.

u. USERAREA. If COMMONSIZE is 60, the User-area field does not exist. If COMMONSIZE is greater than 60, the length of the User-area field (n) is COMMONSIZE minus 60. User-written MESS procedures must be written if this field is to contain significant information.

As previously mentioned, the Common-area header is placed in front of the text of messages exchanged between the MCS and programs using the PARTICIPATION interface. The length of the text is determined by the TEXTSIZE field. For incoming messages and recovered incoming messages, the text is the data received from a terminal. For open notifications, the text is a list of 3-character, Logical Station numbers. No text is associated with attach notifications or detach notifications. For outgoing messages, the Application Program sets the text to the data to be sent to a terminal.

The attach, detach and/or open notifications can be requested by a program using the MCS interface. In this case, GEMCDS writes an MCS-TO-MCS data message with a message type-80 (refer to Burroughs B 1700 Systems Network Definition Language Reference Manual, form 1073715). The text of this data message is a Common-area header. Therefore, a subordinate MCS which expects attach, detach and/or open notifications must be able to handle an MCS-to-MCS data message from GEMCDS in addition to the MCS/Network Controller message types (refer to table 3-1).

Table 3-1

Common-area Header Fields
Containing Valid Information
by MCSTYPE

Field	Written by MCS MCSTYPE (*1)						Written by User Program MCSTYPE (*2) (*3)						
	0	2,4 6	15	21	23	24	0	1	17	18	20	22	25
MSGDESTINATION	X												
LSN	X	X	X				Y	Y					
PGMNR	X												
MTSMSTYPE	X												
SEQNO	X												
NDLTIME	X												
TEXTSIZE	X	X	X	X	X	X							
TERMTYPE	X												
MCSGID	X						Y	Y					
INDEX1	X												
INDEX2	X												
ERROR	X												
FMTERR	X												
MCSTYPE	X	X	X	X	X	X	Y	Y	Y	Y	Y	Y	Y
INPUTADDR	X										W		
RETRYCOUNT	X												
RECOVERYSTATUS	X												
OUTPUTADDR													
USERDATA	U	U					V	V					
TEXT	X		X		X		Y	Y	Y				
CONVERSATIONSTATUS	X												
CONVERSATIONBOJEJ	X					Y	Y						

- 1 An X denotes that the field is set by the MCS and contains valid information; a U indicates that the field is set by the MCS only if procedures for this were specified by the user.
- 2 A Y denotes that the MCS requires the application program to provide valid information in the field; a V denotes that the field is read by the MCS only if procedures were specified by the user.
- 3 A W denotes that the MCS requires the application program to provide information in the field, provided the program employs queue restoration recovery.

MONITOR

MONITOR.

To aid in monitoring, each routine in the MCS is assigned a 3-digit number. The first of the three digits is a code indicating to which module the procedure belongs. The module number 0 (zero) is reserved for MESS procedures. The last two digits identify the procedure. Numbers 90 thru 98 may be assigned to MESS procedures by the MESS programmer.

Corresponding to the last two digits of the procedure number are 99 monitor flags. A given procedure is monitored only if the MCS was generated with the MONITOR parameter set and the corresponding monitor flag has a value of 1. There are two ways to change the settings of monitor flags:

- a. When the MCS is running, by entering a CMF (change monitor flag) Network Control command.
- b. When the MCS is not running, by running the TCL compiler with REGENERATE and setting MONITORTRACEON TRUE or FALSE.

If a procedure is to be monitored, it must contain at least one call on the monitor procedure. The monitor is invoked by a UPL statement of the form:

```
MD.MONITOR(n1, s1, n2, s2);
```

The MD.MONITOR parameters are as follows:

- a. Parameter n1 is the number of the procedure that called the current procedure.
- b. Parameter s1 is a string of up to 30 characters specifying the name of the current procedure.
- c. Parameter n2 is the number of the current procedure.
- d. Parameter s2 is a string of up to 82 characters which contains any information that may be useful for debugging (usually the names and contents of significant data items).

Each time MD.MONITOR is invoked (and the corresponding monitor flag is 1) these parameters are printed in order on one line of the monitor listing, along with the sequence number of the line which called the MONITOR procedure.

Monitor calls are ordinarily the first executable statement of each routine; however, they can be placed anywhere that is useful for debugging.

NETWORK CONTROL COMMANDS.

A Network Control Command (NCC) consists of a signal character, a short mnemonic command code, and in some cases, one or more parameters. Commands are free in form, with words separated by one or more spaces.

Every Network Control Command generates some kind of response. There are three kinds of responses:

- a. Confirmation without data. This response is specified by the user through the (NCC OK RESPONSE statement).
- b. Confirmation with data. For Network Control Commands which request that data be returned, the data itself serves as confirmation that the command was executed.
- c. Rejection. If a command was not successfully executed, a message is returned giving the reason.

In defining the syntax of Network Control Commands, the following conventions are used:

- a. All underlined upper-case words are key words and are required when the functions of which they are a part are utilized.
- b. Upper-case words which are not underlined are optional and may or may not appear in the message.
- c. Lower-case words are generic terms which must be supplied by the user.
- d. When words or phrases are enclosed in brackets [], they may be included or omitted at the user's choice. When words or phrases are enclosed in braces {}, a choice of one of the entries must be made. In both cases, a choice is indicated by vertically stacking the possibilities. When brackets or braces enclose a portion of syntax, but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the syntax to which the following ellipsis applies.
- e. The ellipsis (...) represents the position at which repetition may occur. Repetition is optional. The portion of syntax to be repeated is that which is enclosed within the logically matching pair of brackets or braces whose right-hand member immediately precedes the ellipsis.
- f. The colon character (:) is required when it appears in the syntax, even though it is not underlined. The colon must be preceded and followed by at least one space.

- g. The asterisk (*) is used to denote the location of a signal character. The actual character used in place of the asterisk is determined by the <SIGNAL CHARACTER statement>.
- h. The term "station-specifier" is to be replaced by either of the following:
 - 1) The alphanumeric station identifier assigned to the station in the Station section of NDL.
 - 2) A number which specifies the position that the station definition occupies in the Station section of NDL. (The first station defined is 1; the second station defined is 2; and so on.) This number is also known as the Logical Station Number (LSN).
- i. The term "program-specifier" is to be replaced by either of the following:
 - 1) The alphanumeric program identifier assigned to the program in the Program section of TCL. If a Title statement is given, then this name should be used.
 - 2) A number which specifies the position that the program definition occupies in the Program section of TCL. (The first program defined is 1; the second program defined is 2; and so on. This is independent of the number of copies of any program.)
- j. The term "data base-specifier" is to be replaced by either of the following:
 - 1) The alphanumeric data base identifier assigned to the data base in the DATABASENAME statement of the Program section of TCL.
 - 2) A number which specifies the position that this data base was presented in TCL. The first data base defined is 0; the second data base is 1; and so on.

SECURITY CONTROL COMMANDS.

These commands can be used only if an <ACCESS CONTROL statement> appears in the user source TCL.

SIGN ON (SGN). The SGN command is used to gain access to the system at a station which requires signing on.

Syntax:

* SGN access-code

Example:

3 SGN ABCD

In this example, the ABCD is signed on if it is defined in the <ACCESS CONTROL statement> and is valid for that station.

NCC
cont

SIGN OFF (BYE). The BYE command disconnects a signed-on user from a station. The user should sign off after completing the transaction to ensure that no unauthorized person is able to gain access to the system. BYE cannot be entered from a station which does not require signing on. If the user is attached to a Utility Program (via the EX command) at the time "*BYE" is entered, an implicit HAP of that Utility Program is automatically done by the MCS.

Syntax:

* BYE

ENABLE USER (EUS). The EUS command is used to mark an access code as enabled. The enabled user-ID, with the correct password, can then be used to sign-on.

Syntax:

* EUS access-code

Example:

a EUS ABCD

In this example, the user ABCD may sign on.

DISABLE USER (DUS). The DUS command is used to prevent an access code from being used for logging on.

Syntax:

* DUS access-code

Example:

a DUS ABCD

In this example, the user ABCD is no longer able to sign-on until the access code is enabled again.

PROGRAM CONTROL COMMANDS.

These commands can be used only if the PROGRAMBOJE0J is TRUE.

EXECUTE PROGRAM (EX). The EX command is used to start Assignment, User or Utility Programs, or to attach a station to a Utility Program which is already running. Confirmation of this command merely indicates that MCS communicated ZIP-execute to the MCP; it does not guarantee that the program actually started.

Syntax:

*EX program-name execute-option-list

execute-option-list ::= execute-option / execute-option list,
execute-option /
empty

execute-option ::= LOCK / [charge-number] / user-password

charge-number ::= <integer>

user-password ::= US <usercode> "/" /
US = <usercode> "/"

Semantics:

The execute options are described in detail in the B 1800/B 1700 Systems Software Operation Guide, form number 1068731. If the user-password option was entered during a normal run, it will not be present in the ZIP execute statement, used for recovery.

NOTE

The GEMCOS system does not check for valid usercodes. If the system is unsuccessful performing ZIP execute with the user code, the user must initiate the program by using the SPO.

Examples:

```

a EX PROG/A
a EX PROG/A 12345
a EX PROG/A US AB/CD
a EX PROG/A LOCK US = AB/CD
a EX PROG/A US AB/CD 12345 LOCK

```

HALT APPLICATION PROGRAM (HAP). The HAP command is used to cause an End-of-File condition on the remote file that an Assignment or User Program has open. In the case of Utility Programs, the station at which the HAP command was entered is detached from the program. When the last station detaches itself, an End-of-File condition is sent to the Utility Program. Program-name is optional only for utility programs. If this command is entered during a conversation at a station, the conversation is automatically terminated, and the conversation area is cleared.

Syntax:

* HAP [program-name]

Example:

2 HAP PROG/A
2 HAP

MCS CONTROL COMMANDS.

There are two MCS Control commands: AOK and HLT.

AUDIT OK (AOK). The AOK command is used in response to a message on the Control station or on the console printer of the form:

FILE MISSING - MCSAUDIT/AUDITXXX

It informs the MCS that the requested audit file is available on disk. This command can be entered only at the console printer through an ACCEPT.

Syntax:

* AOK

HALT SYSTEM (HLT). The HLT command brings the data communications system to a stop. This command can be used only if SYSTEMHALT is TRUE.

Syntax:

Ⓒ * HLT KILL

Semantics:

If KILL is not specified, the system comes to an orderly stop, and untransmitted messages are recalled. If KILL is specified, the system comes to an abrupt stop, and messages may be lost.

MESSAGE CONTROL COMMANDS.

There are two MCS control commands: BRC and PQ.

BROADCAST (BRC). The BRC command is used to send a message to other stations in the network. It is available only if MESSAGE BROADCAST is TRUE.

Syntax:

```

          station-specifier
* BRC          ... : message-text
          SPO

```

Semantics:

More than one "station-specifier" may be entered, in which case the message is sent to each station. If no station-specifier is entered, the message is sent to all stations in the network. If SPO is entered, the message is sent to the console printer.

Examples:

```

a BRC : GOOD MORNING
a BRC 3 TD4 : WHAT'S HAPPENING?
a BRC SPO : PLEASE LOAD PROGRAM BLACK/JACK

```


REPORT COMMANDS.

With the exception of RDM, which is controlled by the DATADUMP statement DATA-DUMP, the existence of these commands is controlled by the <STATUS REPORTS statement>.

When examining error statistics it should be kept in mind that:

- a. Counters start at zero each time the MCS is executed.
- b. The MCS increments a counter by the retry limit when the Network Controller reports an error to the MCS, but the Network Controller reports an error only when the retry limit is exceeded. Thus, the number of errors reported in the response to the Network Control Command may be slightly less than the number that actually occurred.

REPORT DATA DUMP (RDM). The RDM command allows access to the contents of MCS data fields.

Syntax:

* RDM PRINT

Semantics:

When PRINT is specified, the report is sent to a system printer and the contents of MCS tables are included. When PRINT is not entered, only nontable information is included. Other report commands are available for displaying table information at a remote station.

REPORT FILE STATUS (RFS). The RFS command returns the following information about a remote file:

- a. File name.
- b. Queue number of the remote file.
- c. Name of the program which opened the remote file.

Syntax:

* RFS [file-name]

Semantics:

If "file-name" is not entered, the status of all remote files is returned.

Example:

2 RFS REMOT1

REPORT PROGRAM STATUS (RPS). The RPS command returns the following information about a program:

- a. Name of the program.
- b. Whether or not the program is running.
- c. Program classification.

Syntax:

* RPS [program-specifier]

Semantics:

If "program-specifier" is not entered, the status of all programs is reported.

Example:

a RPS PROG/A

REPORT PROGRAM COUNTERS (RPC). The RPC command returns the following information about a program:

- a. The number of the program. (See definition of program specifier).
- b. Number of messages sent to the program.
- c. Number of messages received from the program.
- d. The job number of the program, if it is running.

Syntax:

* RPC [program-specifier]

Semantics:

If "program-specifier" is not entered, the status of all programs is reported. If there is more than one copy of a program (MAXCOPIES > 1), statistics are given for all copies of the program including copy number.

Examples:

```
q RPC MY/PROG
q RPC 2
```

REPORT STATION COUNTERS (RSC). The RSC command returns the following statistics about a station:

- a. Number of messages sent.
- b. Number of messages received.
- c. Number of data communications errors.
- d. Number of Network Control Commands affecting the station.
- e. Number of changes made.

Syntax:

* RSC [station-specifier]

Semantics:

If no "station-specifier" is provided, statistics are reported for all stations.

Example:

a RSC TD1

REPORT STATION STATUS (RSS). The RSS command reports the following about a station:

- a. Whether the station is ready, enabled, signed on or attached.
- b. Usage (input, output, or both).
- c. Which program the station is attached to, if any.

Syntax:

* RSS [station-specifier]

Semantics:

If no "station-specifier" is provided, status is reported for all stations.

Example:

a RSS TD1

CHANGE COMMANDS.

The existence of these commands is controlled by the <CHANGE REQUESTS statement> and the <MONITOR TRACE statement>.

CHANGE MONITOR FLAG (CMF). The CMF command is used to enable or disable monitoring of procedures in the MCS.

Syntax:

```
* CMF [flag-number]...:  D
                          N
```

Semantics:

A "flag-number" is the last two digits of a procedure number. More than one flag number may be entered. If no flag number is entered, all monitor flags are changed. N (normal) sets the monitor flag(s) to 0 (zero); D (diagnostic) sets the monitor flag(s) to 1.

Example_1:

```
@ CMF 13 : D
```

Procedures which have "13" as the last two digits will be monitored.

Example_2:

```
@ CMF : N
```

No procedures are monitored.

CHANGE STATION ADDRESS (CSA). The CSA command is used to give the Network Controller a new logical address for a station.

Syntax:

* CSA station-specifier $\begin{matrix} \text{I} \\ \text{O} \end{matrix}$ address

Semantics:

Codes I or O indicate whether the new address applies on input or output.

Example:

a CSA 5 O 1A

Station 5 will have an output address of "1A".

CHANGE STATION DIAGNOSTIC (CSD). The CSD command is used to inform the Network Controller whether or not to use normal diagnostic request logic (as defined in NDL) for a station.

Syntax:

* CSD station-specifier N
D

Semantics:

The code N selects normal request logic; D selects diagnostic logic.

Example:

a CSD 2 D

CHANGE STATION FREQUENCY (CSF). The CSF command is used to assign a new input or output frequency (priority) to a station.

Syntax:

* CSF station-specifier $\begin{matrix} I \\ Q \end{matrix}$ frequency

Semantics:

The codes I and Q specify whether the frequency applies on input or output. "Frequency" must be an integer less than or equal to 255.

Example:

a CSF TD2 I 250

CHANGE STATION MAXIMUM RETRY (CSM). The CSM command is used to give a new value to the number of times the Network Control tries to retransmit a message when there are errors.

Syntax:

* CSM station-specifier retries

Semantics:

"Retries" must be an integer up to two digits long.

Example:

a CSM 1 16

CHANGE STATION QUEUE (CSQ). The CSQ command is used to re-route messages bound for one station to some other station instead.

Syntax:

* CSQ station-specifier-1 station-specifier-2

Semantics:

"Station-specifier-1" is the original station, and "station-specifier-2" is the new station. If "station-specifier-1" and "station-specifier-2" are the same, routing reverts to the original station.

Example:

a CSQ TD1 TD4

Messages which would ordinarily go to station TD1 will now go to station TD4.

CHANGE STATION TRANSMISSION NUMBER (CST). The CST command is used to give a new value to the transmission number of a station.

Syntax:

* CST station-specifier $\begin{matrix} I \\ 0 \end{matrix}$ transmission-number

Semantics:

The codes I and 0 indicate whether the input or output transmission number is to be changed.

Example:

a CST TD1 I 35

FORMATUPDATE (UPD) COMMAND. There is one network control command to update formats. This command is only provided in the advanced and total versions of GEMCOS. The UPD command makes updated formats available to all stations in the MCS network. Formats can be updated using the UPDATEFMT option of the control command or by employing the Format Generator. By either method, updated formats are consequently placed in the MCSFORMATS file. The old version of the format remains available throughout the network until the UPD command is entered. Upon entry of the command, updated formats are available.

Syntax:

* UPD

Semantics:

This request causes all previously-modified formats to be updated and made available to all stations in the network.

Example:

a UPD

AUDIT & RECOVERY COMMANDS.

The existence of these commands is controlled by the **AUDITASSIGNMENT**, **AUDITTRANSACTIONS**, and **RECOVERY** statements of the Program section.

REFRESH COMMAND (REF). The REF command is used to recall the most recent audited output message.

Syntax:

* REF

Semantics:

This request causes the MCS to display the last audited output message for the station. An error is returned if there are no output messages for the station.

Example:

3 REF

CLEAR DISABLED PROGRAM (CLE). The CLE command is used to clear disabled programs and to cause recovery to be initiated.

Syntax:

* CLE <program-specifier>

Semantics:

If after clearing a disabled program recovery does not begin, at least one other program in the data base is still disabled. When all programs are cleared, recovery of the data base initiated.

Examples:

a CLE PROG1
a CLE 2

RECOVER DATA BASE (REC). The REC command is used to initiate recovery on a particular data base or on all data bases declared in TCL.

Syntax:

* REC [<database-specifier>]

Semantics:

If no <database-specifier> is specified, then recovery is initiated for all data bases that are declared to this MCS. Otherwise, recovery is initiated only for the selected data base. If any programs in the data base are disabled at the time of the request, then an error message is displayed on the control station listing the numbers of the disabled program (s). These programs can then be cleared using the CLE command, and recovery is then initiated.

Examples:

```
Q REC
Q REC LIVEDB
Q REC 0
```

RESET BUSY STATUS (RBS) COMMAND. If a station is defined with TRANSACTIONMODE = TRUE, the station can become locked into a busy status. This occurs when a station sends a message to a synchronized recovery program and the program does not respond. Should a station receive Error 106, this indicates that the station has a busy status. The RBS network control command must be entered from a control station or from the SPO.

Syntax:

* RBS station-specifier

Semantics:

Entry of this command causes the station "station-specifier" to be taken out of a busy status. Once the command is processed, the station operator may enter input at the station. If the station is not busy, this command has no effect.

Examples:

```
a RBS STATION2
a RBS 3
```

SECTION 4
MESSAGE ROUTING

GENERAL.

Messages are routed to Application Programs by a GEMCDS MCS by one of three methods, depending on the program classification defined for each Application Program in the <PROGRAM section> of TCL. The three types of classification are Assignment, Utility, and User (refer to ASSIGNMENT PROGRAMS, UTILITY PROGRAMS, and USER PROGRAMS in section 3).

The routing is further defined by the <INTERFACE statement> in TCL. Each program uses one of three interfaces: NONPARTICIPATION, PARTICIPATION, or MCS (refer to NONPARTICIPATION, PARTICIPATION, and MCS in section 3).

Thus seven combinations of program classification and interface are possible: three interfaces for Assignment Programs, three interfaces for Utility Programs, and one interface (PARTICIPATION) for User Programs. The following example lists all seven meaningful TCL descriptions:

PROGRAM X1 UTILITY:	
TITLE = X.	Z OPTIONAL
INTERFACE = NONPARTICIPATION.	
RESIDENCE = DISK.	Z OPTIONAL
PROGRAM X2 UTILITY:	
TITLE = X.	Z OPTIONAL
INTERFACE = MCS.	
OPENMESSAGE = TRUE.	Z OPTIONAL
ATTACHMESSAGE = TRUE.	Z OPTIONAL
RESIDENCE = DISK.	Z OPTIONAL
PROGRAM X3 UTILITY:	
TITLE = X.	Z OPTIONAL
INTERFACE = PARTICIPATION.	Z OPTIONAL
COMMONSIZE = 100.	Z OPTIONAL
TRANCODE = X(1,1).	Z OPTIONAL
OPENMESSAGE = TRUE.	Z OPTIONAL
ATTACHMESSAGE = TRUE.	Z OPTIONAL
DETACHMESSAGE = TRUE.	Z OPTIONAL
RESIDENCE = DISK.	Z OPTIONAL
PROGRAM X4 USER:	
TITLE = X.	Z OPTIONAL
INTERFACE = PARTICIPATION.	Z OPTIONAL
COMMONSIZE = 100.	Z OPTIONAL
TRANCODE = X(1,1).	Z OPTIONAL
OPENMESSAGE = TRUE.	Z OPTIONAL
RESIDENCE = DISK.	Z OPTIONAL
EXECUTE = ONDEMAND.	Z OPTIONAL

```

PROGRAM X5 ASSIGNMENT:
  TITLE = X.                Z OPTIONAL
  INTERFACE = MCS.          Z OPTIONAL
  OPENMESSAGE = TRUE.      Z OPTIONAL
  RESIDENCE = DISK.        Z OPTIONAL

PROGRAM X6 ASSIGNMENT:
  TITLE = X.                Z OPTIONAL
  INTERFACE = NONPARTICIPATION.
  RESIDENCE = DISK.        Z OPTIONAL
  EXECUTE = MANUAL.        Z OPTIONAL

PROGRAM X7 ASSIGNMENT:
  TITLE = X.                Z OPTIONAL
  INTERFACE = PARTICIPATION.
  COMMONSIZE = 100.        Z OPTIONAL
  TRANCODE = X(1,1).       Z OPTIONAL
  OPENMESSAGE = TRUE.      Z OPTIONAL
  RESIDENCE = DISK.        Z OPTIONAL
  EXECUTE = BOJ.           Z OPTIONAL

```

Several general comments apply to the above TCL descriptions:

- a. More than one <TRANCODE statement> is permissible where a <TRANCODE statement> is optionally or mandatorily required.
- b. In a <PROGRAM description> no other <PROGRAM statement> may occur more than once.
- c. The following defaults are in effect:
 - 1) Default for OPENMESSAGE, ATTACHMESSAGE, and DETACHMESSAGE is FALSE.
 - 2) Default for RESIDENCE is CORE.
 - 3) Default for COMMONSIZE is 60.
 - 4) Default for INTERFACE is PARTICIPATION.
 - 5) Default for EXECUTE is MANUAL.

Each of the seven combinations can be used to maximum advantage for a particular program. The following discussion relates how to describe the program to GEMCOS in TCL for each situation.

A user might have a program which was written prior to the acquisition of GEMCOS. Such a program might not require any GEMCOS options other than the capability of remote execution by authorized personnel. This program should be described in TCL as 1. Its program name would be associated in the <ACCESSCONTROL statement> with access keys which are then assigned to authorized persons. Since the program is defined as a Utility Program, it could be executed from any station. Since it used the NONPARTICIPATION interface, it would not have to be recompiled to conform to the GEMCOS Common-area header or to the B 1800/B 1700MCS/ Network Controller interface.

The user may be using CANDE and ODESY, which are MCS programs. Station operators would like the ability to switch between CANDE and ODESY. Without GEMCOS, both CANDE and ODESY must be shut down and brought back up to switch a station. This is very inconvenient to the rest of the station operators. However, if both CANDE and ODESY are defined as X2 programs, GEMCOS, as a supervisory MCS, can switch a station between these two MCS programs on demand from the station without interrupting the rest of the stations. Since sessions with CANDE and ODESY are to be initiated remotely, they are described as Utility Programs. Since both are MCS programs, they must use the MCS interface.

The user may wish to design a program which could be initiated remotely and take advantage of the GEMCOS formatting capability. To support remote execution, it must be a Utility Program. If formatting is to take place, trancodes must be defined and a Common-area header is required. The X3 TCL description would be used.

A situation might arise in which there are several stations, each requiring access to several data bases. It would be undesirable to have one large program handle all the data bases. Conversely, writing several small modular programs for each data base would be more efficient but would require the user to repeatedly execute and halt these programs, an especially awkward and time-consuming process for rarely used programs that process only a few transactions before the next program must be executed. A preferable solution would be to assign trancodes for each data base accessing method, to write modular programs to handle these trancodes, placing these programs in the mix (possibly as disk-resident programs), and to describe a series of X4 programs in TCL. The trancodes would be used to switch messages from any of the stations to any of the programs (and could be used to take advantage of formatting).

The user may have the Remote Job Entry (RJE) package. RJE is an MCS which requires a line to a host system. It would be executed from the supervisory console printer or a card reader, not by the host system. If the host system is occasionally used by some program other than RJE which runs under GEMCOS, the host system has to be included in the remote file opened by GEMCOS. GEMCOS would allow RJE to gain access to the host system if RJE was described as a X5 program. RJE uses the MCS interface and normally communicates with the same station, making it an Assignment Program.

In certain applications the program attaches itself to stations. It is either undesirable or impossible for stations to initiate the program. Perhaps a station is to be dedicated to a program or the station is an output-only device. Programs which determine the message routing via remote file attachment can be accommodated by being described as either an X6 or X7 program. If no GEMCOS capabilities are required, the X6 description suffices. If GEMCOS is to provide functions (formatting, audit, screen wraparound, etc.) for the program, the X7 description is necessary.

SECTION 5
ACCESS CONTROL

GENERAL.

B 1800/B 1700 GEMCOS provides three types of access control:

- a. Requiring a user to sign on with a valid access code before accepting any messages.
- b. Restricting user codes to a specified station or stations.
- c. Restricting each access code to specified program and/or transaction codes.

Access control is defined in three sections of TCL. The <PROGRAM section> defines each program and the transaction codes for each program; the <ACCESS CONTROL statement> defines the access codes and the programs and transaction codes that each access code can use; and the <STATION section> defines the stations in the network and the access codes required to sign on at each station.

A GEMCOS MCS for an on-line system with three programs (a User Program with trancodes UPDATE and INQ, a User Program with trancode EDIT, and a Utility Program) could be generated without security using the following TCL deck:

```
CONTROL = GENERATE, LIST, COMPILE.
GLOBAL:
    CHANGEREQUESTS = TRUE.
    PROGRAMBOJEOJ = TRUE.
    SYSTEMHALT = TRUE.
    MAXTEXTSIZE = 1980.
BEGIN
    PROGRAM PAYROLL USER:
        TITLE = PAYROLL.
        TRANCODE = UPDATE.
        TRANCODE = INQ.
    PROGRAM TEXTEDIT USER:
        TITLE = USERPACK/TEXTEDIT/.
        TRANCODE = EDIT.
    PROGRAM GAME UTILITY:
        TITLE = GAME.
    STATION TD8A:
    STATION TD8B:
    STATION TD8C:
END.
```

The following TCL deck can be used to add user codes to this example system. Each person is required to sign on to GEMCDS with a valid user code before using the system. Once signed on, the user in this example is permitted to use any program or tranocode. The user may sign-on at any station:

```
CONTROL = GENERATE, LIST, COMPILE.
GLOBAL:
  CHANGEREQUESTS = TRUE.
  PROGRAMBOJEOJ = TRUE.
  SYSTEMHALT = TRUE.
  MAXTEXTSIZE = 1980.
BEGIN
  ACCESSCONTROL:
    ACCESSKEY JOE = ALL.      Z ALL MEANS THIS
    ACCESSKEY JIM = ALL.     Z ACCESSCODE IS
    ACCESSKEY TOM = ALL.     Z ALLOWED TO USE ALL
                              Z PROGRAMS AND
                              Z TRANCODES.

  PROGRAM PAYROLL USER:
    TITLE = PAYROLL.
    TRANCODE = UPDATE.
    TRANCODE = INQ.

  PROGRAM TEXTEDIT USER:
    TITLE = USERPACK/TEXTEDIT/.
    TRANCODE = EDIT.

  PROGRAM GAME UTILITY:
    TITLE = GAME.

  STATION TD8A:
    SIGNON = TRUE.
    VALIDACCESSKEYS = ALL.   Z ALL MEANS ALL
                              Z USERCODES CAN
  STATION TD8B:
    SIGNON = TRUE.         Z SIGN ON AT
    VALIDACCESSKEYS = ALL. Z THIS STATION
  STATION TD8C:
    SIGNON = TRUE.
    VALIDACCESSKEYS = ALL.

END.
```

The preceding TCL deck provides three access codes; JOE, JIM, and TOM. All three codes can be used at any station and, once signed on at any station, any program or transaction code can be used. A more restrictive system can be defined. For example, user code TOM could be restricted to trancode EDIT only, and user code JIM to trancode INQ and program GAME, while allowing user code JOE to use all trancodes and programs. The TCL deck with this more restrictive access security scheme would be as follows:

```
CONTROL = GENERATE, LIST, COMPILE.
GLOBAL :
    CHANGEREQUESTS = TRUE.
    PROGRAMBOJEOJ = TRUE.
    SYSTEMHALT = TRUE.
    MAXTEXTSIZE = 1980.
BEGIN
    ACCESSCONTROL:
        ACCESSKEY JOE = ALL.
        ACCESSKEY JIM = INQ, GAME.
        ACCESSKEY TOM = EDIT.
    PROGRAM PAYROLL USER:
        TITLE = PAYROLL.
        TRancode = UPDATE.
        TRancode = INQ.
    PROGRAM TEXTEDIT USER:
        TITLE = USERPACK/TEXTEDIT/.
        TRancode = EDIT.
    PROGRAM GAME UTILITY:
        TITLE = GAME.
    STATION TD8A:
        SIGNON = TRUE.
        VALIDACCESSKEYS = ALL.
    STATION TD8B:
        SIGNON = TRUE.
        VALIDACCESSKEYS = ALL.
    STATION TD8C:
        SIGNON = TRUE.
        VALIDACCESSKEYS = ALL.
END.
```

Finally, certain user codes may be restricted to particular stations. For example, to allow JIM to sign-on only at station TD8A, JOE at TD8A and TD8C, and TOM at all three stations, the TCL deck would be as follows:

```
CONTROL = GENERATE, LIST, COMPILE.
GLOBAL:
  CHANGEREQUESTS = TRUE.
  PROGRAMBOJEOJ = TRUE.
  SYSTEMHALT = TRUE.
  MAXTEXTSIZE = 1980.
BEGIN
  ACCESSCONTROL:
    ACCESSKEY JOE = ALL.
    ACCESSKEY JIM = INQ, GAME.
    ACCESSKEY TOM = EDIT.
  PROGRAM PAYROLL USER:
    TITLE = PAYROLL.
    TRANCODE = UPDATE.
    TRANCODE = INQ.
  PROGRAM TEXTEDIT USER:
    TITLE = USERPACK/TEXTEDIT/.
    TRANCODE = EDIT.
  PROGRAM GAME UTILITY:
    TITLE = GAME.
  STATION TD8A:
    SIGNON = TRUE.
    VALIDACCESSKEYS = ALL.
  STATION TD8B:
    SIGNON = TRUE.
    VALIDACCESSKEYS = TOM.
  STATION TD8C:
    SIGNON = TRUE.
    VALIDACCESSKEYS = JOE, TOM.
END.
```

SECTION 6

MESSAGE FORMATTING

GENERAL.

This section illustrates several practical uses for message formatting. It is not intended as a complete survey of the topic. (Refer to **FORMAT AND FUNCTION STATEMENT LIST** and to **DEVICE SECTION** in section 3 for a description of formatting syntax and semantics.)

In an on-line environment proper message formatting is essential for effective use of the terminal. Properly formatted input and output messages aid readability and assist both the occasional terminal user and the full-time terminal operator. The formatting of input and output messages is therefore a required function that must be performed by each Application Program or by some other part of the system.

When an Application Program formats its own input/output messages, changing or modifying a message format becomes a major problem (e.g., when existing formats are not suitable for a new terminal type). If the formatting is "hard-coded" into the Application Programs, such changes require program patches and recompilation, and can potentially introduce new problems. In addition, this approach requires more memory to store the Formatting code that must be duplicated for each program.

These problems are avoided by using the Generalized Formatting module in the MCS. With this approach, the Formatting code is stored in only one location, the MCS. Thus, messages can be changed without modifying the Application Programs. Since Formatting code is generalized in the MCS and is based on a disk file (MCSFORMATS), format changes can be implemented by merely updating the disk file. The MCS does not have to be modified.

GEMCOS EDITING PHRASES.

The most common GEMCOS editing phrases are described in the following table:

<u>Editing Phrase</u>	<u>Description</u>
ALPHA ITEM TYPE	In the form A<integer>, this phrase moves <integer> bytes from the raw message to the edited message.
INTEGER ITEM TYPE	In the form I<integer>, this phrase moves <integer> bytes from the raw message to the edited message (as does ALPHA ITEM TYPE), and it also verifies that data being entered contains only digits and blanks (no embedded blanks). It also right justifies numbers and inserts leading zeroes where required on input.

<u>Editing Phrase</u>	<u>Description</u>
SKIP FIELD	In the form X<integer>, this phrase inserts <integer> spaces into the message.
EBCDIC STRING	In the form "<character string>", this phrase places a character string into the formatted message on output.
HEXADECIMAL STRING	In the form "<hexadecimal character string>", this phrase is composed of digits 0 thru 9 and letters A thru F. It is used the same way as EBCDIC STRING to insert characters in the message; however, it can be used to insert any character including nonprintable EBCDIC characters. The primary use of this phrase is to insert terminal control characters (e.g., carriage returns, line feeds), many of which are nonprintable.
FUNCTION CALL	In the form T<function name>, <alpha item type>/<integer item type>,<integer>, this phrase converts any string six characters or less to any other string six characters or less. The <function name> is used to specify which function is to be used in the REPLACE operation. The <alpha item type> or the <internal item type> is used to specify the length and type of the external character string and the integer is used to specify the length of the internal character string.
LOCATION SPECIFIERS	In the form @<sign><integer>, this phrase overrides the normal mode of sequential operation of the formatter. It may be required to skip around in the message (e.g., if specific fields are to appear in a different order on the terminal than they come from the program). The @+<integer> skips <integer> bytes forward; the @-<integer> skips <integer> bytes backward.

OUTPUT FORMATTING EXAMPLE.

In this example it is assumed that the purpose of a program is to return information about records in a customer file, and that the program has the following output area and is sending the following data:

	<u>COBOL_Description</u>		<u>Sending_Data</u>
01	SAMPLE-OUTPUT-AREA.		
05	CUSTOMER-NAME	PIC X(30).	THE WILSON FOOD STORES
05	ACCOUNT-NBR	PIC 9(6).	220030
05	TOTAL-AR-BALANCE	PIC ZZZ,ZZZ.99	15,365.50
05	TERMS	PIC X.	3
*	7 = 7 DAYS		
*	1 = 10 DAYS		
*	C = COD		
*	3 = 30 DAYS		
*	05 CURRENT-BALANCE	PIC ZZZ,ZZZ.88	8,420.10
	05 ADDON-PERCENT	PIC ZZ9.	9
	05 OVER-7-DAYS-BALANCE	PIC ZZZ,ZZZ.99	824.10
	05 DATE-LAST-INVOICED	PIC 99/99/99.	08/01/77
	05 OVER-14-DAYS-BALANCE	PIC ZZZ,ZZZ.99	3,281.10
	05 CREDIT-LIMIT	PIC Z(6).	20000
	05 OVER-21-DAYS-BALANCE	PIC ZZZ,ZZZ.99	.00
	05 PRICE-CODE	PIC 9.	2
	05 DELINQUENT-FLAG	PIC 9.	1
*	1 = YES		
*	0 = NO		
	05 OVER-28-DAYS-BALANCE	PIC ZZZ,ZZZ.99	2,840.20
	05 WAREHOUSE	PIC XX.	A1
	05 ITEM-SUBSTITUTE-FLAG	PIC 9.	1
*	1 = YES		
*	0 = NO		
	05 CREDIT-BALANCE	PIC ZZZ,ZZZ.99	.00

The following TCL could be used to format the output messages for a TD700 which has eight lines of 32 characters each:

```
CONTROL = GENERATE, LIST, COMPILE.
GLOBAL:
  CHANGEREQUESTS = TRUE.
  PROGRAMBOJEQJ = TRUE.
  STATUSREPORTS = TRUE.
  SYSTEMHALT = TRUE.
  MAXTEXTSIZE = 1980.
  FORMAT
  OUT700 ( 4"0C0000", % CLEAR SCREEN AND HOME CURSOR
  % LINE 1
    X1,A30,X1,
  % LINE 2
    "ACCT-",I6,X2,"TOTAL",X4,A10,
  % LINE 3
    "TERMS-",A1,X6,"CURRENT",X2,A10,
  % LINE 4
    "ADDON-",A3,X4,"OVER 7",X2,A10,
  % LINE 5
    "L/I-",A8,X1,"OVER 14",X2,A10,
  % LINE 6
    "LIMIT-",A6,X1,"OVER 21",X2,A10,
  % LINE 7
    "PRICE-",I1,X1,"DF-",I1,X1,"OVER 28",X2,A10,
  % LINE 8
    "WHSE-",A2,X1,"SUB-",I1,X1,"CREDIT",X2,A10)).
BEGIN
  PROGRAM SAMPLE USER:
    TITLE = GEMCOSPACK/GEMCOS/SAMPLE.
    TRANCODE = CUSTIN.
  STATION TD7A:
    TRANCODEPOSITION = 1.
  STATION TD7B:
    TRANCODEPOSITION = 1.
  STATION TD7C:
    TRANCODEPOSITION = 1.
  DEVICE TD700:
    STALIST = TD7A, TD7B, TD7C.
    FORMATSOUT:
      OUT700 = ARINFO.
END.
```

Figure 6-1 depicts the formatted output of the data returned from the Application Program (with the format-ID field of the header set to ARINFO) and in turn sent to one of the stations listed under the device "TD700". The MCS uses format OUT700 to format the message.

THE WILSON FOOD STORES			
ACCT-220030	TOTAL		15,365.50
TERMS-3	CURRENT		8,420.10
ADDON- 9	OVER 7		824.10
L/I-08/01/77	OVER 14		3,281.10
LIMIT- 20000	OVER 21		.00
PRICE-2 DF-1	OVER 28		2,840.20
WHSE-A1 SUB-1	CREDIT		.00

Figure 6-1. Sample Formatted Output Data Display

It may be desirable to use functions to convert the data coming from the Application Program to a more readable form. For example, in the preceding format, the TERMS code 3 is not very definitive. It would be better to display something more descriptive like 30 DAY. To do this, the following function is defined:

```
FUNCTION
  TERMS [EXTERNAL:ALPHA,INTERNAL:ALPHA] (
    " 7 DAY": "7",
    "10 DAY": "1",
    "30 DAY": "3",
    "COD   ": "C").
```

This function description converts "7" to "7 DAY", "1" TO "10 DAY", etc. DELINQUENT-FLAG and ITEM-SUBSTITUTE-FLAG displayed as a "1" or a "0" could also be changed to display a Y or N by defining the following function:

```
FUNCTION
  BINARYOUT700 [EXTERNAL:ALPHA,INTERNAL:INTEGER] (
    "Y": "1",
    "N": "0").
```

To take advantage of these functions, the calls must be inserted in the format being used, necessitating the following change to line 3:

```
% LINE 3
  "TERMS-",T(TERMS,A6,1),X1,"CURRENT",X2,A10,
```

Lines 7 and 8 would then read as follows:

```
% LINE 7
  "PRICE-",I1,X1,"DF-",T(BINARYOUT700,A1,1),X1,"OVER 28",X2,A10,
% LINE 8
  "WHSE-",A2,X1,"SUB-",T(BINARYOUT700,A1,1),X1,"CREDIT",X2,A10).
```

The TCL deck now appears as follows:

```
CONTROL = REGENERATE, LIST. % IT IS NOT REQUIRED TO GENERATE AND
        COMPILE % THE MCS SINCE WE ARE ONLY CHANGING FUNCTIONS
        AND FORMATS
```

GLOBAL:

```
  CHANGEREQUESTS = TRUE.
  PROGRAMBOJEOJ = TRUE.
  STATUSREPORTS = TRUE.
  SYSTEMHALT = TRUE.
  MAXTEXTSIZE = 1980.
```

FUNCTION

```
  TERMS [EXTERNAL:ALPHA,INTERNAL:ALPHA] (
    " 7 DAY":"7",
    "10 DAY":"1",
    "30 DAY":"3",
    "COD  ":"C"),
  BINARYOUT700 [EXTERNAL:ALPHA,INTERNAL:INTEGER] (
    "Y":"1",
    "N":"0").
```

FORMAT

```
  OUT700 ( 4"0C0000", % CLEAR SCREEN AND HOME CURSOR
% LINE 1
    X1,A30,X1,
% LINE 2
    "ACCT-",I6,X2,"TOTAL",X4,A10,
% LINE 3
    "TERMS-",T(TERMS,A6,1),X1,"CURRENT",X2,A10,
% LINE 4
    "ADDON-",A3,X4,"OVER 7",X2,A10,
% LINE 5
    "L/I-",A8,X1,"OVER 14",X2,A10,
% LINE 6
    "LIMIT-",A6,X1,"OVER 21",X2,A10,
% LINE 7
    "PRICE-",I1,X1,"DF-",T(BINARYOUT700,A1,1),X1,"OVER 28",
    X2,A10,
% LINE 8
    "WHSE-",A2,X1,"SUB-",T(BINARYOUT700,A1,1),X1,"CREDIT",
    X2,A10).
```

```

BEGIN
PROGRAM SAMPLE USER:
  TITLE = GEMCOSPACK/GEMCOS/SAMPLE.
  TRANCODE = CUSTIN.
STATION TD7A:
  TRANCODEPOSITION = 1.
STATION TD7B:
  TRANCODEPOSITION = 1.
DEVICE TD700:
  STALIST = TD7A, TD7B, TD7C.
  FORMATSOUT:
    OUT700 = ARINFO.

```

END.

Figure 6-2 depicts the formatted output data returned from the Application Programs and forwarded to the stations as a result of the function description and modified TCL deck.

THE WILSON FOOD STORES			
ACCT-220030	TOTAL		15,365.50
TERMS-30	DAY CURRENT		8,420.10
ADDON-	9	OVER 7	824.10
L/I-08/01/77	OVER 14		3,281.10
LIMIT-	20000	OVER 21	.00
PRICE-2	DF-Y	OVER 28	2,840.20
WHSE-A1	SUB-Y	CREDIT	.00

Figure 6-2. Example Formatted Output Data Display for Redefined Forms Function

If two TD830s (having 24 lines of 80 characters each) are now added to the network, a new format is required to take advantage of the larger screen. The larger screen permits more descriptive headers and allows for re-arrangement of the data to make it more readable. A function can be used to convert 1 and 0 to YES and NO rather than the more cryptic Y and N. This requires the defining of a new function:

```
FUNCTION
  BINARYOUT800 [EXTERNAL:ALPHA,INTERNAL:INTEGER] (
    "YES":"1",
    "NO":"0").
```

AND A NEW FORMAT:

```
OUT800 ( 4"0C0000", % CLEAR SCREEN AND HOME CURSOR
% LINE 1
  X17,"CUSTOMER NAME:",X2,A30,4"0D", % 4"0D" IS CARRIAGE RETURN
% LINE 2
  4"0D", % LEAVE BLANK LINE
% LINE 3
  "ACCOUNT NUMBER",X5,I6,X15,"TOTAL A/R BALANCE",X2,A10,4"0D",
% LINE 4
  "TERMS",X14,T(TERMS,A6,1),X15,"CURRENT",X11,A10,4"0D",
% LINE 5
  "ADDON RATE",X9,A3,"Z",X17,"OVER 7 DAYS",X7,A10,4"0D",
% LINE 6
  "DATE LAST INVOICED",X1,A8,X13,"OVER 14 DAYS",X7,A10,4"0D",
% LINE 7
  "CREDIT LIMIT",X7,A3,"",A3,X14,"OVER 21 DAYS",X7,A10,4"0D",
% LINE 8
  "PRICE CODE",X9,I1,2+1, % SKIP OVER SUBSTITUTE FLAG
  X20,"OVER 28 DAYS",X7,A10,4"0D",
% LINE 9
  "WAREHOUSE",X10,A2,2+1, % SKIP OVER DELINQUENT FLAG
  X19,"CREDIT",X13,A10,4"0D",
% LINE 10
  "SUBSTITUTE FLAG",X4,2-24, % SKIP BACK TO GET SUBS FLAG
  T(BINARYOUT800,A3,1),X18,"DELINQUENT FLAG",X4,
  X4,2+12, % SKIP FORWARD TO GET DELINQUENT FLAG
  T(BINARYOUT800,A3,1)).
```

The new stations and the new device type must be defined, resulting in the following TCL deck:

```
CONTROL = REGENERATE, LIST. % IT IS NOT REQUIRED TO GENERATE AND
%   COMPILE THE MCS SINCE WE ARE ONLY CHANGING FUNCTION,
%   FORMAT, STATION AND DEVICE DEFINITIONS.
```

GLOBAL:

```
  CHANGEREQUESTS = TRUE.
  PROGRAMBOJEDJ = TRUE.
  STATUSREPORTS = TRUE.
  SYSTEMHALT = TRUE.
  MAXTEXTSIZE = 1980.
```

FUNCTION

TERMS [EXTERNAL:ALPHA,INTERNAL:ALPHA] (

" 7 DAY": "7",
 "10 DAY": "1",
 "30 DAY": "3",
 "COD " : "C"),

BINARYOUT700 [EXTERNAL:ALPHA,INTERNAL:INTEGER] (

"Y": "1",
 "N": "0"),

BINARYOUT800 [EXTERNAL:ALPHA,INTERNAL:INTEGER] (

"YES": "1",
 "NO " : "0").

FORMAT

OUT700 (4"0C0000", % CLEAR SCREEN AND HOME CURSOR

% LINE 1

X1,A30,X1,

% LINE 2

"ACCT-",I6,X2,"TOTAL",X4,A10,

% LINE 3

"TERMS-",T(TERMS,A6,1),X1,"CURRENT",X2,A10,

% LINE 4

"ADDON-",A3,X4,"OVER 7",X2,A10,

% LINE 5

"L/I-",A8,X1,"OVER 14",X2,A10,

% LINE 6

"LIMIT-",A6,X1,"OVER 21",X2,A10,

% LINE 7

"PRICE-",I1,X1,"DF-",T(BINARYOUT700,A1,1),X1,"OVER 28",X2,A10,

% LINE 8

"WHSE-",A2,X1,"SUB-",T(BINARYOUT700,A1,1),X1,"CREDIT",X2,A10),

OUT800 (4"0C0000", % CLEAR SCREEN AND HOME CURSOR

% LINE 1

X17, "CUSTOMER NAME:",X2,A30,4"0D", % 4"0D" IS CARRIAGE RETURN

% LINE 2

4"0D", % LEAVE BLANK LINE

% LINE 3

"ACCOUNT NUMBER",X5,I6,X15,"TOTAL A/R BALANCE",X2,A10,4"0D",

% LINE 4

"TERMS",X14,T(TERMS,A6,1),X15,"CURRENT",X11,A10,4"0D",

% LINE 5

"ADDON RATE",X9,A3,"Z",X17,"OVER 7 DAYS",X7,A10,4"0D",

% LINE 6

"DATE LAST INVOICED",X1,A8,X13,"OVER 14 DAYS",X7,A10,4"0D",

% LINE 7

"CREDIT LIMIT",X7,A3,"",A3,X14,"OVER 21 DAYS",X7,A10,4"0D",

```

% LINE 8
  "PRICE CODE",X9,I1,2+1, % SKIP OVER SUBSTITUTE FLAG
  X20,"OVER 28 DAYS",X7,A10,4"0D",
% LINE 9
  "WAREHOUSE",X10,A2,2+1, % SKIP OVER DELINQUENT FLAG
  X19,"CREDIT",X13,A10,4"0D",
% LINE 10
  "SUBSTITUTE FLAG",X4,2-24, % SKIP BACK TO GET SUBS FLAG
  T(BINARYOUT800,A3,1),X18,"DELINQUENT FLAG",X4,
  X4,2+12, % SKIP FORWARD TO GET DELINQUENT FLAG
  T(BINARYOUT800,A3,1)).

```

BEGIN

```

PROGRAM SAMPLE USER:
  TITLE = GEMCOSPAC/GEMCOS/SAMPLE.
  TRANCODE = CUSTIN.
STATION TD7A:
  TRANCODEPOSITION = 1.
STATION TD7B:
  TRANCODEPOSITION = 1.
STATION TD7C:
  TRANCODEPOSITION = 1.
STATION TD8A:
  TRANCODEPOSITION = 1.
STATION TD8B:
  TRANCODEPOSITION = 1.
DEVICE TD700:
  STALIST = TD7A, TD7B, TD7C.
  FORMATSOUT:
    OUT700 = ARINFO.
DEVICE TD800:
  STALIST = TD8A, TD8B.
  FORMATSOUT:
    OUT800 = ARINFO.

```

END.

With the new stations properly defined, the Application Program does not have to specify the type of terminal its response is to be sent to. When it sends a message to the MCS with the format-ID set to ARINFO, the MCS uses the appropriate format for the specific device type. If the message is in route to station TD7A, TD7B, or TD7C, it appears on the screen exactly as in the previous example. However, if it is going to TD8A or TD8B, the MCS uses format OUT800 and displays the message depicted in figure 6-3.

CUSTOMER NAME: THE WILSON FOOD STORES			
ACCOUNT NUMBER	220030	TOTAL A/R BALANCE	15,365.50
TERMS	30 DAY	CURRENT	8,420.10
ADDON RATE	9%	OVER 7 DAYS	824.10
DATE LAST INVOICED	08/01/77	OVER 14 DAYS	3,281.10
CREDIT LIMIT	20,000	OVER 21 DAYS	.00
PRICE CODE	2	OVER 28 DAYS	2,840.20
WAREHOUSE	A1	CREDIT	.00
SUBSTITUTE FLAG	YES	DELINQUENT FLAG	YES

Figure 6-3. Example Formatted Output
for TD830 Terminal

INPUT FORMATTING EXAMPLE.

Expanding the example described in OUTPUT FORMATTING EXAMPLE, assume the program needs a message with the following three fields: a 6-character transaction-code field containing CUSTIN; a 6-digit field containing the customer account number; and a 1-digit flag informing the program whether to include aging of the account balance in the response (1 = YES; 0 = NO). Two input formatting requirements are: to build a form on the screen for the operator on which to fill in the required information, and to form the data transmitted by the terminal for the Application Program.

The first objective is met by defining an output format which builds the form on the screen. This format (like all output formats) can be sent to the station by two means: it is sent if an Application Program sends a message to that terminal with the format-ID of the header set to the appropriate format-ID for that format; or the terminal operator can key in the format-ID directly. If it comes from a program, it is filled with the data sent from the program. If the request comes from the station, the format is sent to the station as if a program had sent that format with the data area containing all spaces. Thus, the input form for the TD 700 terminals could be built using the following format:

```
EMPTY 700 C 4"0C0000", Z CLEAR SCREEN AND HOME CURSOR
% LINE 1
  "[CUSTIN]",4"0D",
% LINE 2
  "CUSTOMER ACCOUNT NUMBER",X2,"[",A6,"]",
% LINE 3
  "RECEIVABLE AGING INFO? [",A3,"]",
  4"1200000005000000"). % PUT TERMINAL IN FORMS MODE
  % AND TAB ONCE
```

The DEVICE section for the TD700s must also be changed to read:

```
DEVICE TD700:
  STALIST = TD7A, TD7B, TD7C.
  FORMATSOUT:
    OUT700 = ARINFO.
    EMPTY700 = CSTFRM.
```

Figure 6-4 depicts the display that would be sent by the MCS in response to the forms request (when the terminal operator transmits CSTFRM).

```
[CUSTIN]
CUSTOMER ACCOUNT NUMBER [      ]
RECEIVABLE AGING INFO? [      ]
```

Figure 6-4. Example Input Form for TD700 Terminal

The format for the TD830 terminals is the same except for the form left delimiter and right delimiter which are [and], respectively, on the TD700 and 4"1F" and 4"1E" on the TD 830. Thus, the format can be described as follows:

```
EMPTY800 ( 4"0C0000", Z CLEAR SCREEN AND HOME CURSOR
Z LINE 1
    4"1F","CUSTIN",4"1E0D",
Z LINE 2
    "CUSTOMER ACCOUNT NUMBER",X2,4"1F",A6,4"1E0D",
Z LINE 3
    "RECEIVABLE AGING INFO? ",4"1F",A3,4"1E",
    4"27E6000005000000"). Z PUT TERMINAL IN FORMS MODE
    Z AND TAB ONCE
```

The DEVICE section for the TD830 device would be:

```
DEVICE TD800:
    STALIST = TD8A, TD8B.
    FORMATSOUT:
        OUT800 = ARINFO.
        EMPTY800 = CSTFRM.
```

Thus when the operator transmits CSTFRM from a TD830, the MCS returns the display depicted in figure 6-5.

It is also necessary to format the message from the station for the Application Program. The same format for both the TD700 and the TD830 can be used since both the TD700 and the TD830 send data in the same format (CUSTIN followed by six characters of account number, followed by three characters to indicate whether or not the response is to include aging information).

Since the program expects a 1 or 0 for the aging flag, a function is used to allow the operator to enter YES, Y (or 1 for 1), and to enter NO, N (or 0 for 0). This function is as follows:

```
FUNCTION
    BINARYIN [EXTERNAL:ALPHA,INTERNAL:INTEGER] (
        "YES":"1",
        "Y":"1",
        "1":"1",
        "NO":"0",
        "N":"0",
        "0":"0").
```

Now the format appears as follows:

```
FORMAT
IN (
    A6,I6,T(BINARYIN,A3,1)).
```

```
[CUSTIN]
CUSTOMER ACCOUNT NUMBER [      ]
RECEIVABLE AGING INFO? [      ]
```

Figure 6-5. Example Input Form
for TD830 Terminal

Finally, this format must be added to the FORMATSIN portion of the device description of both the TD700 and the TD830. The TCL is as follows:

```
CONTROL = REGENERATE, LIST. % IT IS NOT REQUIRED TO GENERATE AND
% COMPILE THE MCS SINCE WE ARE ONLY CHANGING FUNCTION,
% FORMAT, STATION AND DEVICE DEFINITIONS.
GLOBAL:
  CHANGEREQUESTS = TRUE.
  PROGRAMBOJEOJ = TRUE.
  STATUSREPORTS = TRUE.
  SYSTEMHALT = TRUE.
  MAXTEXTSIZE = 1980.
```

FUNCTION

```

    TERMS [EXTERNAL:ALPHA,INTERNAL:ALPHA] (
        " 7 DAY": "7",
        "10 DAY": "1",
        "30 DAY": "3",
        "COD   ": "C"),
    BINARYOUT700 [EXTERNAL:ALPHA,INTERNAL:INTEGER] (
        "Y": "1",
        "N": "0"),
    BINARYOUT800 [EXTERNAL:ALPHA,INTERNAL:INTEGER] (
        "YES": "1",
        "NO": "0"),
    BINARYIN [EXTERNAL:ALPHA,INTERNAL:INTEGER] (
        "YES": "1",
        "Y": "1",
        "1": "1",
        "NO": "0",
        "N": "0",
        "0": "0").

```

FORMAT

```

OUT700 ( 4"0C0000", % CLEAR SCREEN AND HOME CURSOR
% LINE 1
    X1,A30,X1,
% LINE 2
    "ACCT-",I6,X2,"TOTAL",X4,A10,
% LINE 3
    "TERMS-",T(TERMS,A6,1),X1,"CURRENT",X2,A10,
% LINE 4
    "ADDON-",A3,X4,"OVER 7",X2,A10,
% LINE 5
    "L/I-",A8,X1,"OVER 14",X2,A10,
% LINE 6
    "LIMIT-",A6,X1,"OVER 21",X2,A10,
% LINE 7
    "PRICE-",I1,X1,"DF-",T(BINARYOUT700,A1,1),X1,"OVER 28",X2,A10,
% LINE 8
    "WHSE-",A2,X1,"SUB-",T(BINARYOUT700,A1,1),X1,"CREDIT",X2,A10,
OUT800 ( 4"0C0000", 5 CLEAR SCREEN AND HOME CURSOR
% LINE 1
    X17,"CUSTOMER NAME:",X2,A30,4"0D", % 4"0D" IS CARRIAGE RETURN
% LINE 2
    4"0D", % LEAVE BLANK LINE
% LINE 3
    "ACCOUNT NUMBER",X5,I6,X15,"TOTAL A/R BALANCE",X2,A10,4"0D",
% LINE 4
    "TERMS",X14,T(TERMS,A6,1),X15,"CURRENT",X11,A10,4"0D",
% LINE 5
    "ADDON RATE",X9,A3,"Z",X17,"OVER 7 DAYS",X7,A10,4"0D",
% LINE 6
    "DATE LAST INVOICED",X1,A8,X13,"OVER 14 DAYS",X7,A10,4"0D",
% LINE 7
    "CREDIT LIMIT",X7,A3,"",A3,X14,"OVER 21 DAYS",X7,A10,4"0D",

```

```

% LINE 8
  "PRICE CODE",X9,I1,2+1, % SKIP OVER SUBSTITUTE FLAG
  X20,"OVER 28 DAYS",X7,A10,4"0D",
% LINE 9
  "WAREHOUSE",X10,A2,2+1, % SKIP OVER DELINQUENT FLAG
  X19,"CREDIT",X13,A10,4"0D",
% LINE 10
  "SUBSTITUTE FLAG",X4,2-24, % SKIP BACK TO GET SUBS FLAG
  T(BINARYOUT800,A3,1),X18,"DELINQUENT FLAG",X4,
  X4,2+12, % SKIP FORWARD TO GET DELINQUENT FLAG
  T(BINARYOUT800,A3,1)),
EMPTY700 ( 4"0C0000", % CLEAR SCREEN AND HOME CURSOR
% LINE 1
  "[CUSTIN]",4"0D",
% LINE 2
  "CUSTOMER ACCOUNT NUMBER",X2,"[",A6,"]",
% LINE 3
  "RECEIVABLE AGING INFO? [",A3,"]",
  4"1200000005000000"), % PUT TERMINAL IN FORMS MODE
  % AND TAB ONCE
EMPTY800 ( 4"0C0000", % CLEAR SCREEN AND HOME CURSOR
% LINE 1
  4"1F", "CUSTIN",4"1E0D",
% LINE 2
  "CUSTOMER ACCOUNT NUMBER",X2,"1F",A6,4"1E0D",
% LINE 3
  "RECEIVABLE AGING INFO? "4"1F",A3,4"1E",
  4"27E60000005000000"), % PUT TERMINAL IN FORMS MODE
  % AND TAB ONCE
IN (
  A6,I6,T(BINARYIN,A3,1)).

```

BEGIN

```

PROGRAM SAMPLE USER:
  TITLE = GEMCOSPAC/GEMCOS/SAMPLE.
  TRANCODE = CUSTIN.
STATION TD7A:
  TRANCODEPOSITION = 1.
STATION TD7B:
  TRANCODEPOSITION = 1.
STATION TD7C:
  TRANCODEPOSITION = 1.
STATION TD8A:
  TRANCODEPOSITION = 1.
STATION TD8B:
  TRANCODEPOSITION = 1.
DEVICE TD700:
  STALIST = TD7A, TD7B, TD7C.
  FORMATSIN:
    IN = CUSTIN.
  FORMATSOUT:
    OUT700 = ARINFO.
    EMPTY700 = CSTFRM.

```

```
DEVICE TD800:
  STALIST = TD8A, TD8B.
  FORMATSIN:
    IN = CUSTIN.
  FORMATSOUT:
    OUT800 = ARINFO.
    EMPTY800 = CSTFRM.
```

END.

Figure 6-6 depicts the filled-in input form. When transmitted, the MCS receives CUSTIN12345 YES. The MCS first checks for a trancode and determines that this message is to be sent to the program GEMCOSPAC/ GEMCOS/SAMPLE because the trancode is CUSTIN. The MCS then formats the message using the format IN since the DEVICE section associates the format IN with the format-ID CUSTIN. After formatting with format IN, the message is CUSTIN0123451. This message is then sent to the Application Program.

```
[CUSTIN]
CUSTOMER ACCOUNT NUMBER [12345 ]
RECEIVABLE AGING INFO? [YES]
```

Figure 6-6. Example Filled-In Input Form

SECTION 7
SYSTEM OPERATION

COMPILING WITH MCSTICL.

For instructions relating to the execution of the TCL compiler, refer to <deck description> under TRANSACTION CONTROL LANGUAGE in section 3.

EXECUTING AN MCS.

Before executing the MCS, the MCSTIC file must be on disk (MCSTIC was created by the TCL compiler). To initiate the MCS, enter:

EX MCSSRC/SOURCE

If a <OBJECT CODE FILE NAME statement> was present in the <GLOBAL section> of the source TCL, enter:

EX <file-ID>

The use of a <FILE statement> following the execute command in an attempt to rename MCSQUEUE is not successful. The MCS always opens always opens the remote file whose name was given in the <QUEUE NAME>. The default value of QUEUE NAME is MCSQUEUE.

When an MCS is run under a system software released prior to 5.1 or when the "C" entry of the name table is blank, a Network Controller must be executed if one is not already running. If a system software release of 6.1 or later is used and there is a nonblank "C" entry in the name table, the Network Controller is automatically executed by the system. (For a description of the system name table entries, refer to B 1800/B 1700 Systems Software Operational Guide, form 1068731.)

Once the GEMCOS MCS begins, the user's application programs and any subordinate MCS programs may be started.

CONSOLE OR CARD READER INPUT.

A Network Control Command may be presented to the MCS at any time by entering an ACCEPT directly to the MCS. For example, the following message entered at the supervisory console makes STATIONA not ready.

<m-n>AX*CSR STATIONA N

The <m-n> is the mix number of the MCS and the asterisk (*) is the signal character. No spaces between AX and the signal character are permitted.

To enter Network Control Commands from a card reader, the operator should ready a card reader with a deck labeled MCSOLICRD and enter:

<m-n>AXCARDS

The MCSOLICRD deck should contain one Network Control Command per card. The signal character must be in card column one.

RECOVERY PROCEDURE.

A GEMCOS MCS which has auditing logic writes audit records into a disk file. This disk audit file is closed:

- a. When it becomes filled.
- b. At End-of-Job.

A new audit file is created after the previous one is filled.

The audit file-ID is MCSAUDIT/AUDITnnn, where <nnn> is a number in the range 0 thru 999 and is incremented for each new audit file.

When the MCS needs an audit file for recovery (other than the audit file it is currently creating) and that file is not currently on disk, it displays the following message on the Control station or console printer:

FILE MISSING - MCSAUDIT/AUDITnnn

While the MCS waits for the backup audit file, it continues to service the network. The operator informs the MCS that the requested backup audit file is available by entering the ADK Network Control Command at the console keyboard.

During recovery, the MCS continues to process messages for programs that are not recovered. If the MCS receives a message which is destined for a program that is being recovered, it sends a message to the originating station indicating that recovery is in progress and that the input message was ignored.

SECTION 8

AUXILIARY PROGRAMS

MCSSIM.

The program MCSSIM makes it possible to test a GEMCDS MCS, including user MESS code, without the presence of a Network Controller or a remote network. Inputs are simulated via a card reader and outputs are listed on a line printer.

The MCS runs in simulation mode when the <SIMULATION statement> specifies a value of TRUE. In simulation mode, the MCS changes the device type of MCSQUEUE from a remote file to a queue file. Hence, input message traffic no longer comes from the network but from any program writing into a queue file labeled MCSQUEUE. MCSSIM is such a program. The source to MCSSIM is MCSIMS.

MCSSIM expects a card file labelled MCSSIMCRD. The format of MCSSIMCRD is closely related to the B 1800/B 1700 MCS/Network Controller interface. The MCS/NC interface consists of a series of message types, each having an explicit function. For a definition of these message types, refer to Burroughs B 1700 Systems Network Definition Language Reference Manual, form 1073715. The user should be familiar with the MCS/Network Controller interface before attempting to use a GEMCDS MCS in simulation mode.

MCSSIM reads cards in a format defined by the MCS/Network Controller interface building a record to be sent to the MCS. The information for each record to be sent to the MCS must be punched beginning in card column 1, extending to column 80, and onto succeeding cards if necessary. Column positions of the cards correspond directly to byte positions of the message types. MCSSIM is able to determine how many cards are required, concatenating the card images into the record to be placed into MCSQUEUE. For example, if a station-status reply was to be sent to the MCS, one card would be punched with 80 characters of information. A second card would be required for the remaining 35 characters (a station-status reply is always 116 characters long). If a message from a program with a text size of 150 characters were to be simulated, three cards would be required. The first card would have the appropriate 50-character message header (with the text-size field set to 150) and the first 30 characters of text. The second card would contain the next 80 characters of text. The third card would have the remaining 40 characters.

One exception exists to the above MCSSIMCRD card format rule. For data message types, the error field, corresponding to card columns 24 and 25, represents 16 bits of information, not two characters of information. The two card columns cannot be used to express all possible combinations of 16 bits. Therefore, card columns 24 and 25 remain blank. The information to be placed in the error field of the record is expressed in the filler field, columns 45 thru 50, as a decimal number representing the desired bit configuration. MCSSIM converts the information in the filler field to a bit string, places the results

into the error field and blanks out the filler field. To simulate a time-out condition, for example, the value 004096 would be punched into the filler field.

If MCSSIM is to simulate messages from a program to the MCS, and that program is using an interface of PARTICIPATION, the Common-area header must be present. For example, suppose that a 10-character message is simulated from a program whose COMMONSIZE is 60. Two cards are required. The first card would have a MCS/Network Controller header in columns 1 thru 50. Columns 51 thru 80 would contain the first 30 bytes of the Common-area header. Columns 1 through 30 for the second card would hold the remaining 30 bytes of Common-area information. The 10 characters of text would follow in columns 31 through 40.

Whether or not the MCS is operating in the SIMULATION mode, it must perform its initialization routines. In order to initialize correctly, the MCS must receive information about the network it is to control or simulate. During initialization, the MCS expects as the first record in MCSQUEUE, a remote file information reply (message type-29).

NOTE

In normal mode, the MCS receives this reply from the Network Controller by issuing a remote file information request. From the remote file information reply, the MCS identifies the appropriate Logical Station Numbers. The MCS expects a station status response (message type-21) for each station in its remote file. Here again, in normal mode, the MCS receives these replies from the Logic Network Controller by issuing station status requests. From the station status replies, the MCS is able to associate Logical Station Numbers with station names.

After the initialization process, the MCS may go through the network restoration procedure. If this is the case, Network Controller change responses (message type-23) may be expected by the MCS. During the initialization process, only change responses and FILE OPEN requests are recognized by the MCS.

Once the MCS ends initialization (or network restoration), the MCS enters its normal message processing mode. Any message can be sent to the MCS for testing purposes. However, care must be taken when the message sent to the MCS causes the MCS to expect a Network Controller response. For example, a valid Change Station Ready command (*CSR In) input to the MCS causes the MCS to issue a change request (message type-22) which must be answered by a Network Controller Change Response message. Somewhere in the MCSSIMCRD deck following the CSR command, there should be a Network Controller change response to the MCS change request. For ease of use, the MCS should not be caused to issue change, recall or remove requests.

EXAMPLE SIMULATION CARD DECK.

In this example (see figure 8-1) a network with one station, LSN01, is simulated. The first card of MCSSIMCRD is a remote file information reply specifying that there is one station in the remote file. The second and third cards are a station status reply describing LSN01 as being ready, enabled, having a retry count of 255, etc. The fourth card simulates a report program status command from LSN01. Cards five and six simulate a FILE OPEN request by a program named PGMA. A message from PGMA to LSN01 is simulated with card seven.

NOTE

PGMA has included a Common-area header of 60 bytes.

```
?DATA MCSSIMCRD
29000000000000000000100000000001      001000
210010001LSN1      11344022500      020207      07
2550000000001000000000000000000000003
010001000400100000000000 00000000000000000044000000=RPS
10300000000000000000000000      PGMA      0 00002      001CRMT1
      001
000001006800200000000000 000000000000000000440000000010      0008
      HI THERE
?END
```

Figure 8-1. Example Simulation Card Deck

MCSFIX.

MCSFIX provides the patching capabilities of the UPL compiler. Thus a source-code file may be patched, listed and/or sequenced without having to be compiled. In particular, this program is intended for use with MCSGTS, should it be necessary to release a patch notice.

The file names used by MCSFIX correspond to those of the UPL compiler:

- a. Disk input source file is labeled SOURCE.
- b. Disk output source file is labeled NEWSOURCE.
- c. Card input patch file is labeled CARDS.

MCSFIX always expects the input card file and creates the output disk file (although the disk file may be closed with purge). Records in CARDS fall into one of two categories: control cards and patch cards. Control cards, identified by a dollar sign (\$) in column one, specify MCSFIX options. Patch cards and cards without a dollar sign in column 1 are used to create new lines of source or to replace existing lines.

Columns 73 through 80 are reserved for optional sequence numbers. When sequence numbers exist in either of the two input files, they should be in ascending order; otherwise, an error is reported. However, when an error is detected, the run is not aborted even though results are unpredictable (but possibly of value).

The following is a list of available options and their actions:

<u>Option</u>	<u>Action</u>
LIST	This option creates a listing of the output disk file. If a sequence number is specified in columns 73 through 80, the list starts at that number or, if it does not exist, at the next highest number. By default, LIST is set.
MERGE	When this option is set, MCSFIX expects the file labeled SOURCE, merges records from the file CARDS and creates the output file (refer to option to save the output file). A record from CARDS replaces a record in SOURCE when both have the same sequence number. A record from CARDS is inserted when there is no record in SOURCE with a matching sequence number. (To delete records in SOURCE, refer to VOID option.) When MERGE is not set, an input disk file is not expected, and the output disk file is created directly from the input card file. The control card specifying MERGE must precede any patch cards and may not be reset. By default, the MERGE option is off.

<u>Option</u>	<u>Action</u>
NEW	This option causes the output disk file to be saved (closed with lock). The control card with NEW specified must precede any patch cards and may not be reset. By default, the NEW option is off.
NO	This option turns off or reverses the effect of any options which follow it on the same control card. This is different than the UPL compiler (in which NO applies only to the option immediately following it).
SEQ <ssssssss> + IIII	This option causes sequence numbers starting with ssssssss and incrementing by IIII to be applied to the output disk file (and printer listing if LIST is set). IIII may not exceed 9999. NO SEQ terminates any sequencing taking place. Sequence numbers of records in CARDS are always relative to the old sequence numbers in SOURCE regardless of whether or not the sequencing of the output disk file is specified.
VOID <nnnnnnnn>	The VOID option causes records from SOURCE not to be included in NEWSOURCE (i.e., it deletes them). <nnnnnnnn> must be present in columns 73 through 80. If <nnnnnnnn> is not present, the record in the input disk file with the sequence number <nnnnnnnn> is voided. When <nnnnnnnn> is present in columns 7 through 14, all records with sequence numbers <nnnnnnnn> through <nnnnnnnn>, inclusive, are voided.

To execute MCSFIX, the following control cards are used:

- a. ? EXECUTE MCSFIX
- b. ? FILE SOURCE NAME <input-file-identifier>;
- c. ? FILE NEWSOURCE NAME <output-file-identifier>;
- d. ? DATA CARDS
- e. (control and patch cards)
- f. ? END

MCSRECALL.

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 an audit file. There are two ways to retrieve messages from the audit file; by simply recalling the last <n> messages or by recalling messages by time of day. Only one program needs to be declared to the TCL compiler. The trancode(s) used must be declared within the Program section of the Recall Program. The Global section statement necessary to declare a Recall Program follows.

```
RECALLPROGRAM = <identifier>.
```

The following rules must be adhered to when defining the attributes of a Recall Program.

- a. The program must be declared as a User Program.
- b. The COMMONSIZE statement must be absent or set to a value of 60 (default value).
- c. The program must not use the auditing capability.

Included on the GEMCOS release tape is a fully functional message Recall Object Program called MCSRECALL. The user need only declare this program to TCL.

Prior to initially executing the supplied Recall Program, the user must make the following modifications:

- a. The external file name associated with the MCSTIC file must be the same as that associated with the MCSTIC file of the MCS.
- b. The external file name of the MCSREM remote file must be a remote file of the Network Controller. Also, the Number of Stations (NST) attribute must be set to the number of stations requested by the remote file.

See Appendix E for a summary of all files contained in MCSRECALL.

The syntax of a recall message (as expected by MCSRECALL) is as follows:

```

<recall message> ::= [declared trancode from TCL]
                    <period>
                    <recall primitive>
                    <period>

<recall primitive> ::= TIME <slash> <time recall list> /
                    LAST <slash> <last recall list>

<time recall list> ::= <recall source>
                    <time of day range>
                    <date specification>
                    <message type>
                    <message destination>

<last recall list> ::= <recall source>
                    <integer>
                    <message type>
                    <message destination>

<recall source> ::= (<station number>) <slash> /
                    <station mnemonic> <slash> /
                    <empty>

<time of day range> ::= <time of day> <dash> <time of day> /
                    <time of day>

<date specification> ::= <slash> ON <date> / empty

<message type> ::= <slash> <message type identifier> /
                    <empty>

```


MCSRECALL
cont

<message destination> ::= <slash> PRINTER / empty
<station number> ::= <integer>
<station mnemonic> ::= <identifier>
<time of day> ::= [4-digit time of day ranging from
0000 to 2400]
<date> ::= <month> <slash> <day> <slash> <year>
<message type identifier> ::= INPUT / IN /
OUTPUT / OUT /
IO
<month> ::= [integer ranging from 1 to 12]
<day> ::= [integer ranging from 1 to 31]
<year> ::= [2-digit integer specifying year]

If <recall source> is empty, then the recalled messages are for the station entering the request; otherwise, they are for the station name or number requested.

If <message type identifier> is INPUT, only the specified input messages are recalled; if OUTPUT, only the specified output messages are recalled; if I/O, then both the input and the corresponding output messages are recalled. If <message type identifier> is empty, only output messages are recalled.

The recall message by time-of-day allows the user to specify a time range in which to indicate the messages to be recalled. If a date is also specified, the messages for that date are recalled if the corresponding audit file or files are on disk.

If LAST is specified, the last <n> messages requested are recalled. If there are fewer than <n> messages to recall, then the number of messages found is recalled.

If PRINTER is selected from <message destination>, then all the recalled messages are sent to the system printer instead of to the requesting station.

Examples: (IRC is the user's TCL-defined tranocode for the Recall Program)

IRC.TIME/1200.	Z Recall the output messages stamped with Z time 1200 for this station.
IRC.TIME/1200-1215/IO.	Z Recall both the input and output messages Z between 1200 and 1215 for this section.
IRC.TIME/0800-1600/ON 03/31/79 /IO/PRINTER.	Z Recall both the input and output messages Z between 8 AM and 4 PM on March 31, 1979 Z and send them to the system printer.
IRC.TIME/(3)/0900-0930 /INPUT.	Z Recall the input messages initiated at Z station 3 between 9 AM and 9:30 AM and Z send them to the requesting station.
IRC.TIME/LSN2/1000- 1045/ON 04/12/79 / IO/PRINTER.	Z Recall both the input and output messages Z from station LSN2 between 10 AM and 10:45 Z AM on April 12, 1979 and send them to the Z system printer.
IRC.LAST/10.	Z Recall the last 10 output messages for Z this station.
IRC.LAST/(2)/5/IO.	Z Recall the last 5 input messages and Z associated output messages from station 2 Z and send them to the requesting station.
IRC.LAST/50/PRINTER.	Z Recall the last 50 output messages for Z this station and send them to the system Z printer.
IRC.LAST/LSN3/100/ IO/PRINTER.	Z Recall the last 100 input messages and Z associated output messages from station Z LSN3 and send them to the system printer.

SECTION 9

RECOVERY

GENERAL.

B 1800/B 1700 Series GEMCOS provides a broad range of recovery capabilities within the TCL. This gives the user the ability to analyze application-oriented needs and to select the recovery options required.

In this manual, recovery means that a failure has occurred, thereby interrupting the ability to continue normal transaction processing until some corrective action is taken. The recovery process is considered to be the re-establishment of normal processing.

Throughout this section, assume that the user wants to minimize exposure to situations wherein file and/or data base reloads are required following a failure. In such situations, 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 can be costly to the user and is based directly upon several factors. The main factors follow:

- a. Data base size.
- b. Length of time since last data base dump.
- c. Transaction volumes since last data base dump.
- d. Transaction-throughput activity.

Further, assume that the user desires to free user programmers from recovery concerns as much as is possible, since 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 1800/B 1700 Series GEMCOS is to make the recovery process present, yet virtually invisible, to application programs. In return, the user is expected to follow several straightforward programming conventions for normal processing. These conventions vary slightly, depending upon the level of recovery desired.

Although the MCS is "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 II). 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" program or data base. This option is the default option for every program declared in TCL. It may also be specified in the Program section with the following TCL syntax:

RECOVERY = NONE.

When the system or the program fails, no attempt is made to reprocess any messages that may have been lost at the time of failure. This is true whether or not the MCS is auditing messages for that program.

TYPES OF RECOVERY.

The lowest level of recovery, queue restoration, is for programs which do not share a data base with any other program. Queue restoration places the responsibility for determining if recovery is required on the user application program. The user application program must also direct the MCS to begin recovery with a certain specified message. When the MCS receives a request from a program for queue restoration, it resends all succeeding audited messages to that program.

The next level of recovery, data base recovery, is for Application Programs which share a data base with other programs. The only recovery responsibility placed on the User Program is to save certain information from the MCS-supplied header in a place where it can be accessed by the Restart Program (defined later in this section). In DMS II programs, this is normally the restart data set. The other recovery requirement for such a User Program is that the program must notify the MCS when a DMS II abort occurs. When the MCS determines that recovery is needed (at BOJ, because a program has failed or because a program notified the MCS that a DMS II abort occurred), the MCS executes the Restart Program. After all messages are received from the Restart Program, the MCS begins recovery of each program in the data base that requires it. The order of recovery is oldest message first.

The most sophisticated form of recovery, synchronized recovery, is similar to data base recovery. In this type of recovery, the MCS recovers each message in the order it updated the data base when the message was originally processed. To accomplish this, the MCS assigns a Data Base Sequence Number (DBSN) to each output message from the program. When recovery is required, the order of message recovery is in DBSN order rather than oldest input message first. Also, during synchronized recovery, the MCS performs an output message analysis to minimize duplicate output messages at the station.

QUEUE RESTORATION RECOVERY.

When a user application program receives a transaction from the MCS in the course of normal processing, it should check the MCS-INPUT-ADDR field in the Common-area header. If this field is nonzero and this program may later need to be recovered, the data in the MCS-INPUT-ADDR field must be saved. When recovery is required, the user must return the data in the MCS-INPUT-ADDR field to the MCS. If the MCS-INPUT-ADDR field is zero, the transaction was not audited and the user application program should ensure that this transaction does not cause any updates unless this program does not use recovery.

The MCS performs queue restoration for programs which have a "RECOVERY = QUEUERESTORATION." statement in the Program section of TCL. This recovery is initiated when the user application program sends a message to the MCS with the Common-area header field MCS-TYPE set to 20. Also, this program should set the Common-area header field MCS-INPUT-ADDR to the value it contained in the last completed transaction. This value would have been saved as previously described in the course of normal processing. The user application program should then ignore all transactions from the MCS until it receives a message with MCS-TYPE set to 21. Next, the user application program should send a message to the MCS with MCS-TYPE set to 22. This "handshaking" process ensures that no extraneous messages are processed by the program. Finally, the MCS passes all the recovered transactions back to the application program and normal transactions can again be sent as they are received. The Application Program can determine when recovery is complete by checking the MCS-RECOVERY-STATUS field in the Common-area header. A value of zero indicates that this message is a normal message.

When more than one program needs queue restoration concurrently, the MCS passes the messages to the programs in the order they were placed on the audit file in one pass of the audit file. The MCS makes no attempt to reproduce the exact sequence of transactions and/or data base updates which occurred during normal processing.

During queue restoration, the MCS does not perform any output message analysis. The MCS passes all recovered output received from the Application Programs to the stations.

NONSYNCHRONIZED AND SYNCHRONIZED DATA BASE RECOVERY.

The following discussion assumes a basic knowledge of the B 1800/ B 1700 Systems Data Management System II (DMS II) Reference Manual, form number 1089794. Programs that were declared in the Program section of TCL with a DATABASENAME statement fall into one of two categories. The first category consists of programs that were declared with the TCL statement "RECOVERY = DATABASE.". This is nonsynchronized data base recovery and henceforth shall simply be called data base recovery. The second category consists of programs that were declared with the TCL statement "RECOVERY = SYNCHRONIZED.". This is synchronized data base recovery and henceforth shall be called synchronized recovery.

REMOTE FILE OPENS.

When a data base or synchronized recovery program opens its remote file, the first message sent to it has the Common-area header field MCS-TYPE set to a value of 23. The first three bytes of the message text area contain the program number, the next three bytes contain the multi-copy number, and the next twelve bytes contain the date/time stamp. In user application programs using DMS II, this information must be saved to place in the restart data set. See figure 9-1 for a recommended data set definition, and figure 9-2 for a remote file record definition.

```
RESTARTAREA RESTART DATA SET (
  GEMCOS-LITERAL           ALPHA (6);
  GEMCOS-PGM-NBR           NUMBER (3);
  GEMCOS-MULTI-NBR         NUMBER (3);
  GEMCOS-DATE-TIME         NUMBER (12);
  GEMCOS-DATA              NUMBER (9))
X
POPULATION = 100;
X
RESTARTSET ORDERED SET OF RESTARTAREA
  KEY IS (
    GEMCOS-LITERAL,
    GEMCOS-DATE-TIME,
    GEMCOS-PGM-NBR,
    GEMCOS-MULTI-NBR);
```

Figure 9-1. Recommended DMS II Restart
Data Set Definition

```

01  GEMCOS-REMOTE-FILE-RECORD.
    05  GEMCOS-COMMONAREA-HEADER.
        10  MCS-LSN                PIC 9(4).
        10  FILLER                 PIC 9.
        10  MCS-SEQ-NO             PIC 9(6).
        10  MCS-NDL-TIME          PIC 9(7).
        10  MCS-TEXT-SIZE         PIC 9(4).
        10  MCS-TERM-TYPE        PIC 9(2).
        10  MCS-MSG-ID            PIC X(6).
        10  MCS-INDEX-1          PIC 9(2).
        10  MCS-INDEX-2          PIC 9(2).
        10  MCS-FMT-ERR           PIC 9(2).
        10  MCS-TYPE              PIC 9(2).
        10  MCS-INPUT-ADDR        PIC 9(9).
        10  MCS-RETRY-COUNT       PIC 9.
        10  MCS-RECOVERY-STATUS   PIC 9.
        10  MCS-OUTPUT-ADDR      PIC 9(9).
        10  FILLER                PIC 9(2).
        10  MCS-USER-AREA         PIC X(N1). (*5)
    05  MCS-TEXT                  PIC X(N2). (*6)
    05  MCS-TYPE-23-MESSAGE REDEFINES MCS-TEXT.
        10  GEMCOS-HEADER-PGM-NBR PIC 9(3).
        10  GEMCOS-HEADER-MULTI-NBR PIC 9(3).
        10  GEMCOS-HEADER-DATE-TIME PIC 9(12).
        10  FILLER                PIC X(N3). (*7)

```

Figure 9-2. Recommended Remote File Record Definition

5 N1 is the length of the user portion of the common area header; if there is no user portion, then MCS-USER-AREA should not be specified.

6 N2 is the length of the user's text.

7 N3 is N2 minus 18.

Upon receipt of this message, the following COBOL code should be executed in the DMS II user application program:

```
MODIFY RESTARTSET AT
  GEMCOS-LITERAL      = "GEMCOS"                AND
  GEMCOS-DATE-TIME    = GEMCOS-HEADER-DATE-TIME AND
  GEMCOS-PGM-NBR      = GEMCOS-HEADER-PGM-NBR   AND
  GEMCOS-MULTI-NBR    = GEMCOS-HEADER-MULTI-NBR
ON EXCEPTION
  CREATE RESTARTAREA
  MOVE "GEMCOS"        TO GEMCOS-LITERAL
  MOVE GEMCOS-HEADER-DATE-TIME TO GEMCOS-DATE-TIME
  MOVE GEMCOS-HEADER-PGM-NBR   TO GEMCOS-PGM-NBR
  MOVE GEMCOS-HEADER-MULTI-NBR TO GEMCOS-MULTI-NBR
  MOVE 0                TO GEMCOS-DATA.
```

TRANSACTION PROCESSING.

A transaction is any message in which the MCS-TYPE is set to the value zero. A DMS II application program must do two things in addition to processing each input transaction. First, it checks the MCS-INPUT-ADDR field of the Common-area header. If the field is zero, the program should not do any updates to the data base, as this transaction has not been audited and will, therefore, not be recovered. Second, if the MCS-INPUT-ADDR field is nonzero, it must be saved in the restart data set (i.e., "MOVE MCS-INPUT-ADDR TO GEMCOS-DATA."). If MCS-TYPE is set to zero, updates to the restart data set are permitted.

Output messages are then sent to the MCS according to the following:

- a. MCS-TYPE should be set to zero on the last message.
- b. MCS-TYPE should be set to one on any intermediate transactions.

A minimum of one output message must be sent to the MCS for each audited input transaction.

END-OF-JOB.

The user application programs should be prepared for the receipt of a message from the MCS with MCS-TYPE set to 24. This message instructs the program to close the data base. If the data base close is successful, the program should send the MCS a response message having MCS-TYPE set to 25. The MCS then sends an End-of-File (EOF) indicator, and the program should go to End-of-Job (EOJ). If a program abort is detected when the data base is closed, the program should send the MCS a message with MCS-TYPE set to 20. If an Application Program closes its remote file and/or goes to EOJ at any other time, the MCS considers the program to be aborted. Application programs may not delete the restart data set record before proceeding to End-of-Job.

PROGRAM ABORT.

When an Application Program associated with a data base aborts (i.e., closes its remote file before receiving an EOF or is DSeJ), the MCS marks that program disabled and stops accepting messages from stations

for any program using that data base. When an operator or the network administrator enables the program (using the CLE Network Control Command) from the Control station or the console, after taking any corrective action, the MCS executes the Restart Program. After the Restart Program sends the MCS the necessary recovery information, the aborted program is re-executed by the MCS. All transactions not reflected on the data base are re-sent to the program they were originally sent to. The MCS then returns to normal processing.

RECOVERY PROCESSING.

When a user application program detects a DMS II abort, it should send the MCS a message with MCS-TYPE set to 20. The program should then ignore all messages from the MCS until it receives a message with MCS-TYPE set to 21. When a user application program receives a message with MCS-TYPE set to 21 (whether or not it has sent a type-20 message), it should execute the following coding sequence:

```
BEGIN-TRANSACTION NO-AUDIT RESTARTAREA
ON EXCEPTION
    <exception handling code> .
END-TRANSACTION NO-AUDIT RESTARTAREA
ON EXCEPTION
    <exception handling code> .
```

This code sequence clears the DMS II abort flag prior to the initiation of recovery. An ABORT exception at BEGIN TRANSACTION or END TRANSACTION, can be ignored; processing does continue. Finally, the program should send a message to the MCS with MCS-TYPE set to the value 22. The MCS then sends the program the transactions that were rolled back from the data base. When recovery is finished, the MCS begins sending normal transactions as they are received. The application program can determine when recovery is complete by checking the MCS-RECOVERY-STATUS field in the common area header. A value of zero indicates that this message is a normal message.

RECOVERY AFTER SYSTEM FAILURE.

If a MCS terminates abnormally (i.e., a DS or DP condition arises or a HLT KILL was done) or the MCP fails and a Clear/Start is necessary, the MCS, when re-executed, executes the Restart Program and checks the audit file to determine if any transactions are not reflected in the data base. If there are transactions that are not reflected in the data base, the trancodes' program or programs are automatically executed by the MCS. All unprocessed transactions are sent to the corresponding programs. Normal processing begins when all recovered messages are sent.

DATA BASE RECOVERY (NONSYNCHRONIZED).

Data base recovery performs a queue restoration on all Application Programs in the data base needing recovery. Data base recovery does not attempt to duplicate the order of updates to the data base or to perform any output analysis. The only difference between data base recovery and queue restoration after the initialization is that data base recovery does not allow messages to be entered from stations for any program using the data base until all programs using the data base have finished recovery.

SYNCHRONIZED RECOVERY.

This form of recovery was defined earlier in this section under Types of Recovery. DMS II rolls back a data base in time to remove 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, so that input transactions which were removed from the data base are "recycled" for application processing. Input transactions which have been successfully audited by the MCS prior to failure, but which have not yet reached the point where they are passed to an application program for processing are requeued. Transactions falling into the "recycle" category 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, the MCS attacks the more complex problem of re-establishing 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 concurrently, and thus, each transaction could result in multiple concurrent updates to the data base. This feature becomes extremely important when multiple transactions which result in access to common data are asynchronously flowing through the system at the same time. Synchronized recovery allows failures to occur without Application Programs and terminal operators needing to have any knowledge of them. 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, an analysis of input transactions recycled due to a data base rollback is performed by the MCS on application-generated output to eliminate duplicate transmissions. To accomplish this, the MCS directs the Network Controller to return a "good results" reply when a primary output message is received at the station.

The MCS ensures that a successful, totally synchronized recovery has occurred. Yet, under certain circumstances, the MCS may lose output messages or send duplicate output messages to stations. Extraneous updates to the data base, the most critical cases, do not occur. If output messages are lost, the MCS notifies the terminal operator of the extent of that loss. If duplicate output messages are sent, the MCS informs the terminal operator that a list of possible duplicate messages follows. After sending these messages, the MCS informs the terminal operator that the possibly duplicate messages are finished. In both cases, the number of messages lost or duplicated is usually quite small.

RECOVERY-RELATED CONVENTIONS.

The conventions which must be followed by the user to ensure a synchronized recovery are detailed in the following paragraphs.

CONVENTIONS FOR A "RECOVERY = SYNCHRONIZED" SPECIFICATION.

- a. A User Program must claim transaction-related resources prior to updating any of them. Thus, all necessary data management records should be locked before entering the transaction state. In many situations, this might entail merely locking a particular node within a data hierarchy.
- b. A User Program must not unlock any transaction resources until the transaction is complete. Thus, the DMS II END-TRANSACTION can be allowed to unlock data management records.
- c. A User Program must send messages to stations according to the following protocol:
 - 1) "Secondary outputs" are sent first. These are outputs which are passed to the MCS with the MCS-TYPE field of the Common-area header set to 1. The value 1 signifies to the MCS that the User Program is not done with the current transaction and wishes to send more output.
 - 2) 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". The primary output of a transaction is a message sent by the User Program with a value of 0 in the MCS-TYPE field of the Common-area header. The value 0 signifies to the MCS that the User Program is finished with the current transaction. The MCS assigns the Data Base Sequence Number (DBSN) at this time. The primary output may be delivered to a station other than the originating station, or it may have a text length of zero. In the latter case, no output message is delivered to the station and the MCS audits an output message of zero length.
- d. 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 transaction state.

The proper sequence of events for a User Program which has synchronized recovery is outlined in the following basic logic flow example.

Example:

```
INITIALIZE.
  OPEN DATABASE.
  READ MCSQUEUE AT END  STOP RUN.
  IF MCS-TYPE NOT = 23  STOP RUN.
  MODIFY RESTARTSET AT
    GEMCOS-LITERAL      = "GEMCOS"                AND
    GEMCOS-DATE-TIME   = GEMCOS-HEADER-DATE-TIME AND
    GEMCOS-PGM-NBR     = GEMCOS-HEADER-PGM-NBR   AND
    GEMCOS-MULTI-NBR   = GEMCOS-HEADER-MULTI-NBR
  ON EXCEPTION
    CREATE RESTARTAREA
      MOVE "GEMCOS"                TO GEMCOS-LITERAL
      MOVE GEMCOS-HEADER-DATE-TIME TO GEMCOS-DATE-TIME
      MOVE GEMCOS-HEADER-PGM-NBR   TO GEMCOS-PGM-NBR
      MOVE GEMCOS-HEADER-MULTI-NBR TO GEMCOS-MULTI-NBR
      MOVE 0                        TO GEMCOS-DATA.
  PERFORM PROCESS THRU PROCESS-EXIT UNTIL EOF = 1.
  STOP RUN.
PROCESS.
  READ MCSQUEUE AT END  MOVE 1 TO EOF
                        GO TO PROCESS-EXIT.

  IF MCS-TYPE = 24
    <SEND "TYPE 25" MESSAGE>
    GO TO PROCESS-EXIT.
  IF MCS-TYPE = 21
    <PROCESS & SEND "TYPE 22" MESSAGE>
    GO TO PROCESS-EXIT.
  <LOCK DATA BASE RECORDS TO BE UPDATED>
  BEGIN-TRANSACTION NO-AUDIT RESTARTAREA.
  <MOVE RESTART INFORMATION TO RESTART AREA>
    *
    *
    *
  <UPDATE DATA BASE>
  <SEND SECONDARY OUTPUTS>
  END-TRANSACTION AUDIT RESTARTAREA.
  <SEND PRIMARY OUTPUT>
PROCESS-EXIT.
EXIT.
```

RESTART PROGRAM.

Every system of programs (data base) that has synchronized recovery must have a Restart Program defined. The Restart Program is defined in the Program section of 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 executed by the MCS as the first step in its synchronized recovery mechanism. The Restart Program retrieves all the data base restart data of Application Programs that will be involved in the recovery mechanism, sorts it, and passes it to the MCS. The Restart Program then goes to EDJ. The MCS also executes the Restart Program if the previous run of the MCS terminated abnormally.

GEMCDS supplies a documented COBOL source file of a sample Restart Program on the 4.00 release tape which is 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 can be patched by the user to replace the naming conventions of the restart data set used by GEMCDS with the user's data names. This sample source also contains documentation on the values of the Common-area header field MCS-TYPE. The only required change to this source is the data base name, and possibly the remote file name. Otherwise, this source can be compiled and integrated into the system.

RECOVERY CYCLE.

The GEMCDS recovery cycle can be initiated in any of the following ways:

- a. By the Network Control Command REC.
- b. When a user program passes the abort indicator back to the MCS (a message with MCS-TYPE set to 20).
- c. When the MCS is restarted after it has abnormally terminated or after the operating system has abnormally terminated.
- d. When an Application Program using recovery abnormally terminates (with the CLE Network Command).

In the first three cases, recovery is initiated automatically when the situation is recognized by the MCS. In the fourth case, recovery is indicated after the aborted program is cleared by the operator.

Any transaction which is recycled to a User Program during recovery is flagged with MCS-RECOVERY-STATUS of the Common-area set to one of the following values:

- a. VALUE 0 - The system is not in recovery mode.
- b. VALUE 1 - The system is in recovery mode caused by an Application Program abort.
- c. VALUE 2 - The system is performing an archival recovery.
- d. VALUE 3 - The system is in recovery mode caused by a Clear/Start or an abnormal termination of the MCS.

In addition, a transaction which causes a User Program to abnormally terminate is resubmitted with the Common-area header field MCS-RETRY-COUNT incremented by one. The MCS always re-submits an aborted transaction. The User Program must decide how to process the transaction.

Figure 9-3 shows the sequence of events at a Control station during recovery after a User Program has abnormally terminated. Messages prefixed by (u) are messages that were entered at the station by the user. Messages prefixed by (s) are messages that were sent by the MCS to the station. Any other station that attempted to input transactions during recovery (and was ignored) is informed that recovery is finished. A station that is idle during recovery is not informed that recovery is done.

```
(u) TRN1 <data>
(s) **ERROR 435 126 : UNEXPECTED CLOSE FROM PGM - <nnn>
(s) ?? 127 TO INITIATE RECOVERY, CLEAR PGM - <nnn>
(u) *CLE <nnn>
(s) $OK$
(s) ?? 108 : BEGIN RECOVERY OF DATABASE - <mmm>
(s) ?? 109 : END RECOVERY OF DATABASE - <mmm>
```

Figure 9-3. User Program Abnormal Termination Recovery Cycle

ARCHIVAL RECOVERY.

It might become necessary to reconstruct the sequence of events performed upon the data base as initiated by the data communications network over a period of many days. Conditions necessitating such action might arise if DMS II reconstruction fails or if a program bug contaminates the data base. The archival recovery mechanism is the means by which this reconstruction of events is accomplished.

Archival recovery uses the MCS-created audit files from the original processing of transactions. To initiate archival recovery, the MCS is executed with switch 1 set to 1 (SW1 = 1). After initialization, the MCS displays the following messages on the display console:

- a. ?? 110 : GEMCOS ARCHIVAL RECOVERY
- b. ?? 111 : ENTER ARCHIVAL SPECIFICATIONS

The following controlling information must be supplied through the display console.

- a. The name of the data base to be recovered.
- b. The list of all programs in the data base that are to be recovered or, if all programs are to be recovered, then the word "ALL".

This information must be supplied in free format according to the following syntax:

```
<archival recovery request> ::= RECOVER DATABASE
                               <data base specifications>

<data base specifications> ::= <data base name>
                               <program list>

<program list>                ::= <program name> /
                               <program name> , <program list> /
                               ALL
```

Before initiating the archival recovery mechanism, the operator must insure that the data base on disk reflects a known "good" state. The MCS may require previously dumped audit files which also need to be loaded by the operator. If they are not present, the MCS prompts the operator to load the necessary files. The archival recovery mechanism recovers only one data base per archival run. If more than one data base needs to be recovered, separate archival runs are necessary.

During archival recovery, no messages generated by Application Programs are released to the stations and no input is allowed from any station. When archival recovery is finished, the MCS goes to End-Of-Job. All messages from the MCS during archival recovery are sent to the display console. If one or more programs are disabled at the time of archival recovery, they are automatically enabled before archival recovery begins.

Figure 9-4 shows a typical sequence of events and console traffic for an MCS archival recovery run. Messages prefixed by a (u) are generated by the operator, and messages prefixed by an (s) are generated by either the MCS or the MCP. When the archival recovery process is finished, the data base is recovered and the MCS can then be re-executed in "normal" mode to restart on-line processing.

```
(u) EX GEMCOS/MCS; SW1 = 1
(s) GEMCOS/MCS = 1234 BOJ.
(s) % GEMCOS/MCS : ?? 110 : GEMCOS ARCHIVAL RECOVERY
(s) % GEMCOS/MCS : ?? 111 : ENTER ARCHIVAL SPECIFICATIONS
(u) 1234AX RECOVER DATABASE TESTDB ALL
(s) % GEMCOS/MCS : ?? 115 : BEGIN ARCHIVAL RECOVERY
(s) % GEMCOS/MCS : ?? 108 : BEGIN RECOVERY OF DATABASE - <nnn>
(s) % GEMCOS/MCS : ?? 109 : END RECOVERY OF DATABASE - <nnn>
(s) % GEMCOS/MCS : ?? 116 : END ARCHIVAL RECOVERY
(s) GEMCOS/MCS = 1234 EOJ.
```

Figure 9-4. MCS - Archival Recovery Cycle

CONCLUSION.

Many different types of MCS-TYPE messages are passed to and from the MCS during the recovery cycle. The following brief table provides the user with a quick, easy reference.

Table 9-1
Recovery Control Messages

MCS-TYPE Value	Written by	Definition and Usage
15	MCS	Sent by MCS to Restart Program requesting restart data be retrieved
17	Restart Program	Sent by Restart Program to MCS. Contains restart data information for all programs in the data base
18	Restart Program	Sent by Restart Program to MCS whenever a fatal error is encountered
20	User Program	Sent to MCS by Application Program whenever program encounters a DMS II program abort
21	MCS	Sent to Application Program by MCS informing program that recovery is about to be initiated
22	User Program	Sent to MCS by Application Program in response to type 21 message
23	MCS	Sent to Application Program by MCS at FILE OPEN. Contains pertinent restart data information.
24	MCS	Sent to Application Program by MCS instructing program to close the data base
25	User Program	Sent to MCS by Application Program in response to type-21 message

Burroughs has accepted responsibility for playing 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 10

STATION TYPES

GENERAL.

B 1800/B 1700 GEMCDS provides interface to four different types of stations: AP300, MT600, Routeheader, and standard. This attribute is specified in a STATION statement within the STATION section of the TCL. Depending on the type declared, other parameters may also be required.

at the time of the failure may converse with programs that are not required in the recovery process. Once the recovery process is complete, all stations and programs are free to converse.

The AP300 is a free-standing matrix printer capable of operating with the data communications interface of a host system. B 1000 GEMCDS provides the interface for on-line operations between the AP300 and application programs.

The AP300 has various options which may be programmatically loaded into the printer from a host program. The AP300 has the capacity to send various status conditions to the computer. B 1000 GEMCDS intercepts the status condition messages and displays them at the network controller station. Optionally, as specified by the user, GEMCDS forwards the status message to the program to which the AP300 is attached. (This capability only exists if the AP300 is attached to an assignment program.)

MT600.

The modular terminal, (sometimes called the soft terminal), is a sophisticated forms processing system. When a form is created on the terminal, a program is also created to direct the processing of the form. After processing, the form is stored either in a file at the terminal or in the host processor. If the form is stored in the host processor, it is loaded down-line at a later time. By user specification, the program directs either some or all data fields of the form to the host processor. Similarly, the form receives either some or all data fields. In addition to the memory allocated for the form and the programs, an additional memory area is available for direct communication with the host processor. This area is referred to as the command message area.

To sort out this complex array of communications possibilities, special headers have been designed to precede messages and indicate the nature of the text being sent. With the exception of command message area (C/M Area) messages, all messages from the modular terminal have these headers, and all messages to the terminal are preceded by them. Furthermore, all messages other than continued C/M area and forms messages must be followed by a special trailer.

The GEMCDS interface to the modular terminal does not include formatting the input or output of the terminal. The interface capability serves to do the following:

- a. Determine the type of message that was input from the terminal.

- b. Extract the trailer and header from the input message.
- c. Determine the type of message to be output to the terminal.
- d. Add the trailer and header (extracted from the input message) to the output message.

A one-character field in the common area header serves to identify all message types, directed to or originating from application programs.

PROCESSING INPUT FROM THE MTS.

The GEMCDS system examines all messages from the MTS and determines whether a header is present in each. When no header is present, a zero is placed in the message type field of the common area header. Zero indicates the message is from the common area. When the header is present, the message type is taken from the header and placed in the message type field. The header and trailer are then removed from the message text. Auditing, routing and other types of processing proceed normally.

PROCESSING OUTPUT TO THE MTS.

The GEMCDS system constructs the header for the response to the MTS by employing the message type that was received from the MTS in the common area header. Before the GEMCDS system constructs headers for return messages, it validates the message type. GEMCDS discards any message, having an invalid header identification, and sends an error message to the control station. When the message type is zero (0) in the message from the MTS, no header or trailer is sent with the message returned to the MTS. Otherwise, the header and trailer are created and sent with the MTS.

MODULAR TERMINAL SYSTEM MESSAGE TYPES.

The following list provides the valid message types that are in the common area header of messages, passed between programs and the GEMCDS system.

- 0 = Send or receive to command message area
- 2 = Send or receive total form definition
- 2 = Send or receive first buffer of total form definition
- 3 = Send or receive condensed form
- 3 = Send or receive first buffer of condensed form
- 4 = Send or receive all data fields
- 6 = Send or receive selected data fields
- 7 = Recovery point message
- 8 = Last continuation buffer (forms only).
- 8 = Intermediate continuation buffer (forms only).

SECTION 11

CONVERSATIONAL FEATURE

GENERAL.

The conversational feature is a unique method of communication between a participating program and a terminal. Conversations involve three system elements: the station, the program, and the MCS. Participating program messages typically consist of the GEMCDS header and message text. In conversational messages, conversation text is embedded between the header and the message. Conversation text is only updated by the program. When a program sends a message, the MCS removes the conversation text from the message and stores it in the conversation area of memory. As new text is returned from the station, the MCS returns the conversation text to the message before delivering it to the program.

Through this procedure, the program is relieved from declaring tables or allocating memory for the conversation text; instead, these tasks are assumed by the MCS. The MCS allocates a conversation table. Each conversation area is an element of the table. The MCS allocates the conversation table according to the maximum number allowed in the system simultaneously. Areas are only used and brought into memory as required.

Stations may only converse with one program at a time. Conversely, programs may communicate with several stations simultaneously. Normally, program messages can still be sent to a station that is conversing with a program. Each conversation is assigned one conversation area. There may be many conversational programs and conversational stations in the TCL. The MCS needs to maintain only one copy of the conversation table. This process eliminates duplicate information and reduces memory usage.

TCL SPECIFICATIONS.

To create the conversational capability, three statements are used: CONVERSATIONLIMIT statement, CONVERSATIONSIZE statement, and CONVERSATIONAL statement.

CONVERSATIONLIMIT STATEMENT.

This statement determines the maximum number of stations that may converse concurrently. The statement is entered in the GLOBAL section. Any conversation initiated which exceeds the limit established with this statement causes an error. The initial message of the conversation which caused the error is not sent to the intended destination; instead, it is returned to the program with an error status in the message header. The program may either periodically attempt to send the initial conversation message until a conversation area is available (i.e., until another conversation has been terminated) or, it may send a message notifying the station operator of the temporary shutout.

CONVERSATIONSIZE STATEMENT.

The conversation size varies among programs. The size is established for each, using this statement. When several programs have conversational capabilities, the largest conversation size among them is applied as the conversation size for all conversation areas.

The conversation area allocated by the MCS may be much larger than the conversation sizes specified for certain programs. However, the MCS only provides the number of bytes required by each program. Areas must be allocated at the start of running a program; they cannot be allocated as required. Afterward, the areas may be used and assigned as required.

NOTE

The message size declared must be large enough for conversation messages. Conversation messages consist of a header, conversation text, and message text. The MAXTEXTSIZE statement is used to establish the message size.

CONVERSATIONAL STATEMENT.

This statement is used to assign the conversational capability to stations. It is assigned to a station being defined by assigning the value TRUE to the statement; assigning the value FALSE excludes a station from the conversational capability. If a program attempts to communicate with a nonconversational station, an error occurs. (For further information about this statement, refer to the STATION section of this publication.)

All three statements, just described in this section, are declared in the TCL as follows:

- a. For the GLOBAL section (optional):

CONVERSATIONLIMIT = <nn>.
(nn represents a two-digit number)

- b. For the PROGRAM section:

- 1) CONVERSATIONSIZE = <nnn>.
(nnn represents a three-digit number)
- 2) INTERFACE = PARTICIPATION.
- 3) AUDITOUTPUT = TRUE.
(this attribute must be declared for recovery of conversations)

- c. For the STATION section (optional):

CONVERSATIONAL = TRUE.
(TRUE is the default value)

PROCEDURES FOR CONVERSATIONAL PROGRAMS.

It is the task of all conversational programs to indicate the beginning and the end of a conversation. They also include the conversation text in the message. Two fields have been added to the message header which enable the program to perform these tasks: the CONVERSATIONBOJEOJ field and the CONVERSATIONSTATUS field.

Once a program receives a message, the value of the CONVERSATIONSTATUS field indicates whether the path is clear for the message, a conversation is in progress at the station, or the message has caused an error. A definition for each value follows:

- 0 - Path is clear; either no conversation is in progress, or the station is nonconversational.
- 1 - Conversation is in progress at the station. The value of the CONVERSATIONBOJEOJ field indicates whether another program is conversing with the station.
- 2 - Error. The maximum number permitted for simultaneous conversations has been exceeded. The last program message is returned (without the audit) to the program.
- 3 - Error. Program attempts to initiate a conversation with a nonconversational station. The message is returned to the program.
- 4 - Error. Program attempts to converse with a station which is currently conversing with another program. The message is returned to the program, in error.
- 5 - Error. A nonconversational program initiates a conversation. The message is returned to the program.

The program designates the appropriate value for the CONVERSATIONBOJEOJ field in the message header. This field has one of two values in messages directed to a station. Definitions for both values follow:

- 0 - No conversation in progress. MCS expects message text immediately after the message header.
- 1 - Conversation is in progress. Conversation text follows the header and precedes the output text. It is stored in the conversation area. However, if the program is being audited, both the conversation text and output text are audited.

In messages directed to a program (i.e., from a station), the CONVERSATIONBOJEOJ field has two possible values. Definitions for both follow:

- 0 - Unoccupied. This indicates the station is available for conversation.
- 1 - Occupied. The program that is currently conversing with the station receives this value in each conversation message from the station.

A recommended remote file record and working storage definition is presented in figure 11-1.

**RECOMMENDED REMOTE FILE RECORD
AND WORKING STORAGE DEFINITION**

```

01  REMOTE-FILE-RECORD.
    05  COMMON-HEADER.
        10
        10  MCS-LSN-NBR                PIC 9.
        .                               PIC 9(3).
        .
        .
        10  MCS-OUTPUT-ADDR            PIC 9(9).
        10  MCS-CONVERSATION-STATUS    PIC 9.
        10  MCS-CONVERSATION-BOJEOJ    PIC 9.
    05  MCS-USER-AREA                  PIC X(N1).  (*8)
    05  TEXT-AREA                      PIC X(N2).  (*9)
    05  TEXT-WITH-CONV REDEFINES TEXT-AREA.
        10  CONVERSATION-TEXT          PIC X(N3).  (*10)
        10  REGULAR-TEXT               PIC X(N4).  (*11)

WORKING-STORAGE SECTION.
01  WS-INPUT                          PIC X(N4).  (*11)
    .
    .
    .

01  WS-OUTPUT                          PIC X(N5).  (*12)
    .
    .
    .

01  WS-CONVERSATION-AREA              PIC X(N3).  (*10)
    .
    .
    .

```

Figure 11-1. Recommended Remote File Record
and Working Storage Definition

-
- 8 N1 represents the length of the user portion of the common area header. If there is no user portion, the MCS-USER-AREA should not be given.
 - 9 N2 represents the length of the user text.
 - 10 N3 represents the length of the user's conversation text.
 - 11 N4 represents N2 minus N3.

The proper sequence of events for a user program with conversational capabilities is outlined in the following basic-logic-flow example. (See figure 11-1 for remote file record and working storage definitions.)

Example:

```
PROCESS.
  READ MCSQUEUE AT END MOVE 1 TO EOF
  GO TO PROCESS-EXIT.
  IF MCS-TYPE = 24.
    <SEND "TYPE 25" MESSAGE>
    GO TO PROCESS-EXIT.
  IF MCS-TYPE = 21
    <PROCESS & SEND "TYPE 22" MESSAGE>
    GO TO PROCESS-EXIT.
  IF MCS-CONVERSATION-STATUS > 1
    <PROCESS CONVERSATION ERROR>
    GO TO PROCESS-EXIT.
  IF MCS-CONVERSATION-STATUS = 0    OR
    (MCS-CONVERSATION-STATUS = 1 and
     MCS-CONVERSATION-EOJEOJ = 0)
  *   NO CONVERSATION TEXT IN MESSAGE
  *   MOVE TEXT-AREA TO WS-INPUT
  ELSE
  *   CONVERSATION TEXT IN MESSAGE
  *   MOVE CONVERSATION-TEXT TO WS-CONVERSATION-AREA
  *   MOVE REGULAR-TEXT TO WS-INPUT.
  <START PROCESS>
  .
  .
  .
  IF MCS-CONVERSATION-BOJEOJ = 0 and
    MCS-CONVERSATION-STATUS = 1
    <NO CONVERSATION ALLOWED>
    MOVE WS-OUTPUT TO TEXT-AREA
    <OR BUILD OUTPUT IN TEXT-AREA>
  ELSE
    <MAY INITIATE OR CONTINUE CONVERSATION>
    <TO BEGIN OR CONTINUE CONVERSATION:>
    MOVE WS-CONVERSATION-AREA TO CONVERSATION-TEXT
    MOVE WS-OUTPUT TO REGULAR-TEXT
    <OR BUILD OUTPUT DIRECTLY IN TEXT-AREA>
    MOVE 1 TO MCS-CONVERSATION-BOJEOJ

    <NO CONVERSATION OR TO END A CONVERSATION>
    MOVE WS-OUTPUT TO TEXT-AREA
    MOVE 0 TO MCS-CONVERSATION-BOJEOJ.

  <SEND OUTPUT>

PROCESS-EXIT.
  EXIT.
```

To avoid reducing recovery efficiency, programs must terminate completed conversations, particularly at stations that continually receive nonconversational messages.

RECOVERY OF CONVERSATIONAL PROGRAMS.

The AUDITOUTPUT statement of a conversational program must be declared with the value TRUE in the program section in order to recover conversation text after a system failure. Conversation text is recovered through audited output messages.

Any conversations which are in progress when a system failure occurs are recovered. New conversations are temporarily excluded from participating stations until the recovery process is complete. However, nonconversational messages are permitted at all stations and programs while the system is being recovered. Stations that are not conversing at the time of the failure may converse with programs that are not required in the recovery process. Once the recovery process is complete, all stations and programs are free to converse.

SUMMARY.

The conversational feature provides various benefits. The number of file accesses are reduced by storing data that is repeatedly used by programs. Conversation areas are brought into memory as required. The number of copies of a program does not affect the area size allocated. Also, stations conversing with programs are only assigned one conversation area. These imposed limitations eliminate duplication of information and reduce the space required to store station information for a program. Finally, access to certain transactions and programs can be controlled by restricting the number of stations that are assigned the conversational capability.

APPENDIX A
SUMMARY OF TCL SYNTAX

GENERAL.

This appendix describes the TCL syntax using "railroad track" diagrams. It is provided as an alternative to the Bachus-Naur form used to describe the TCL syntax in section 3. The following rules apply to the syntax diagrams:

- a. Any path traced along the forward direction of the arrows produces syntactically valid TCL source code.
- b. Any "bridge" over a digit may be traversed a maximum number of times specified by the digit. If the digit is followed by an asterisk (*), the path must be crossed at least once.
- c. Upper-case letters in the syntax diagrams indicate key words which are literally in the statement.
- d. The (darkened square) indicates the end of a TCL source file.
- e. Lower-case letters, words and phrases enclosed within angle brackets (< and >) are either references to an intermediate diagram or syntactic variables, which represent information to be supplied by the user.
- f. The (undarkened square) indicates the end of an intermediate diagram.
- g. A <file-ID> is a B 1800/B 1700 file identifier. Refer to B 1800/B 1700 Systems Software Operational Guide, form 1068731.
- h. A <remote file-ID> is a 10-character identifier defined in the FILE section of NDL.
- i. A <station name identifier> is a 10-character identifier defined in the STATION section of NDL.
- j. All other TCL identifiers may contain alphanumeric characters. Identifiers have no intrinsic meaning. They are used to name access codes, programs, trancodes, messages, formats and functions. The <access code identifiers> and <message identifiers> may not exceed 6 characters. <Trancode identifiers> may not exceed 10 characters. <Program name identifiers>, <format identifiers> and <function identifiers> should not exceed 30 characters.
- k. A <string> is any number of characters enclosed within quotes. A <string> must begin and end on the same TCL source record.

APPENDIX A (cont)

- l. An <external string> or an <internal string> is a <string> of 6 characters or less.
- m. An <EBCDIC unit string> is a <string> with exactly one character.
- n. A character is A thru Z, 0 thru 9, or any valid special character. A character is not enclosed within quotes.
- o. An integer is a string of digits (0 thru 9).
- p. For a definition of a <UPL DECLARATION statement>, <UPL DEFINE statement>, <UPL FILE statement>, <UPL DYNAMIC DECLARATION statement> or <UPL PROCEDURE statement>, refer to the UPL Reference Manual, form 1067170.

Figures A-1 thru A-20 depict the TCL syntax using "railroad track" diagrams.

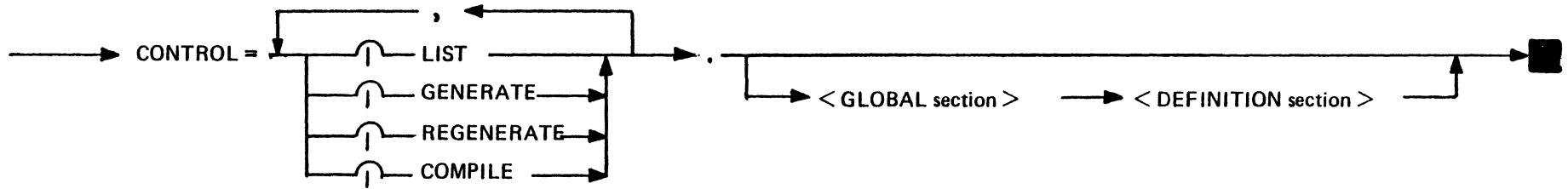


Figure A-1. Syntax of <MCSTCL deck description>

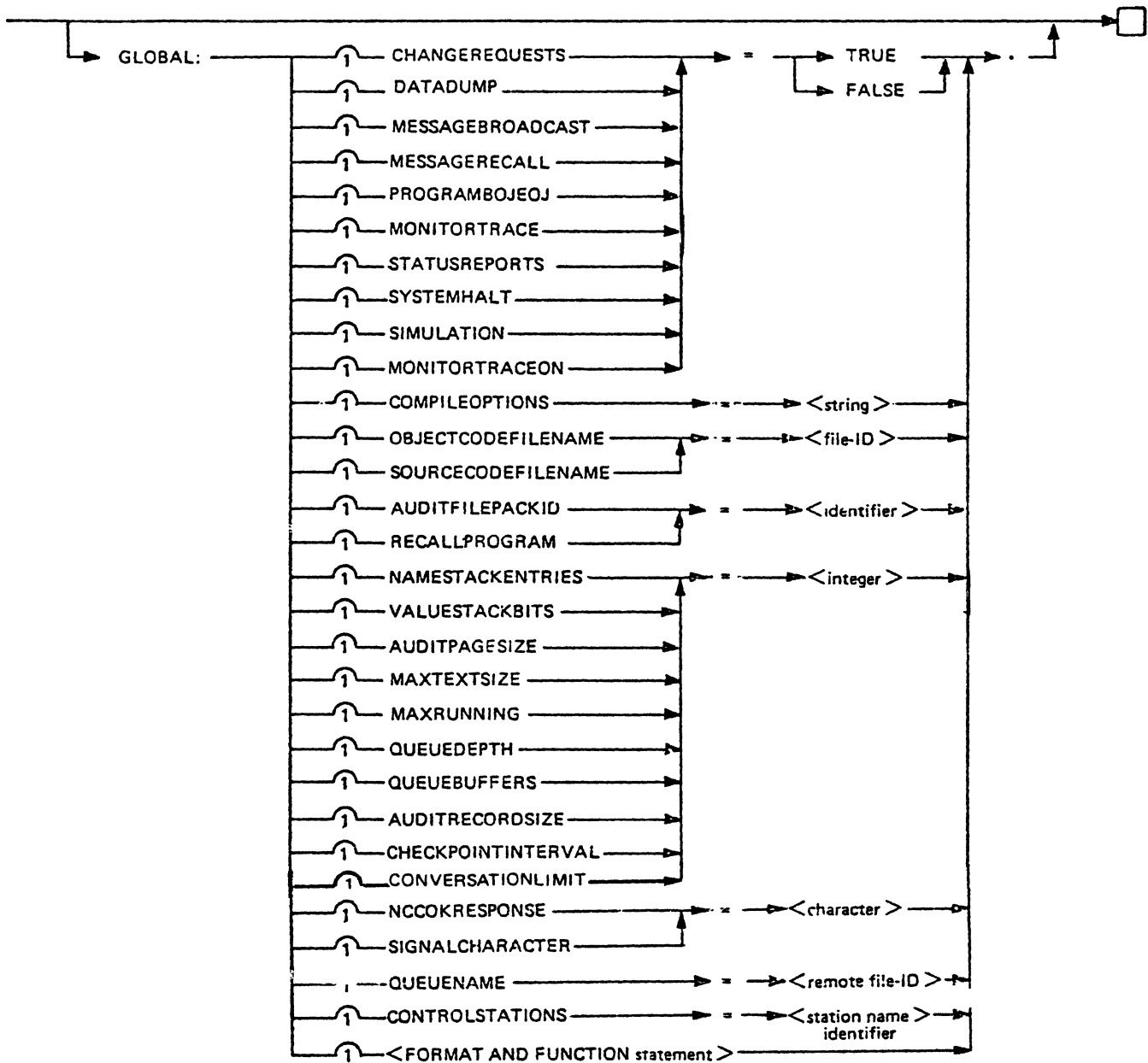


Figure A-2. Syntax of <GLOBAL section>

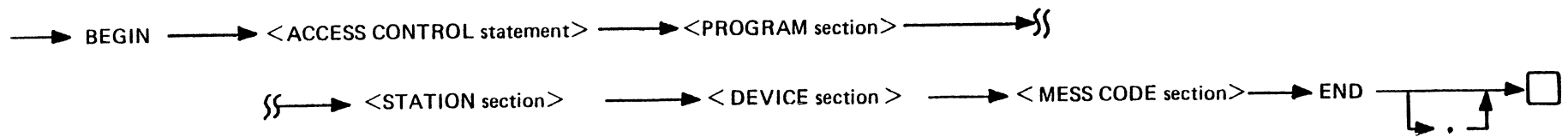


Figure A-3. Syntax of <DEFINITION section>

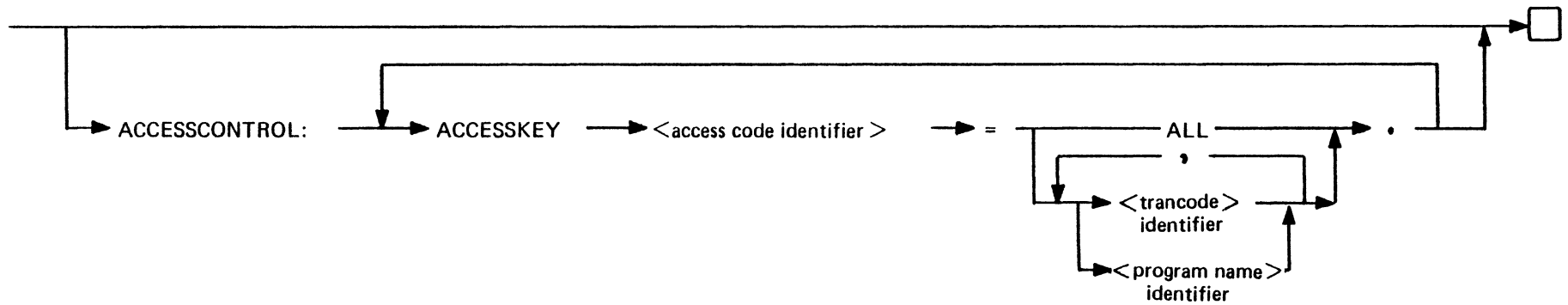


Figure A-4. Syntax of <ACCESS CONTROL statement>

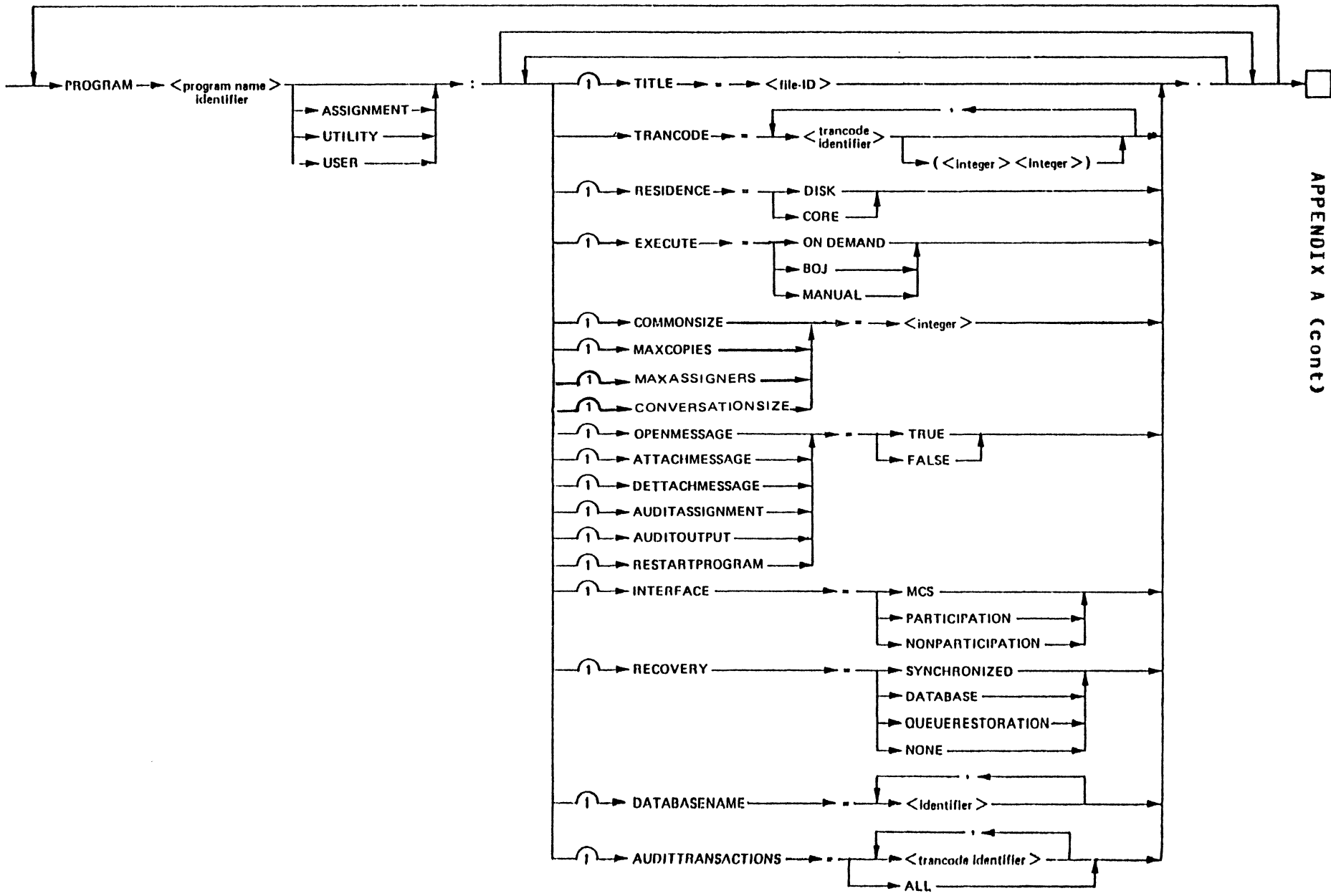


Figure A-5. Syntax of <PROGRAM section>

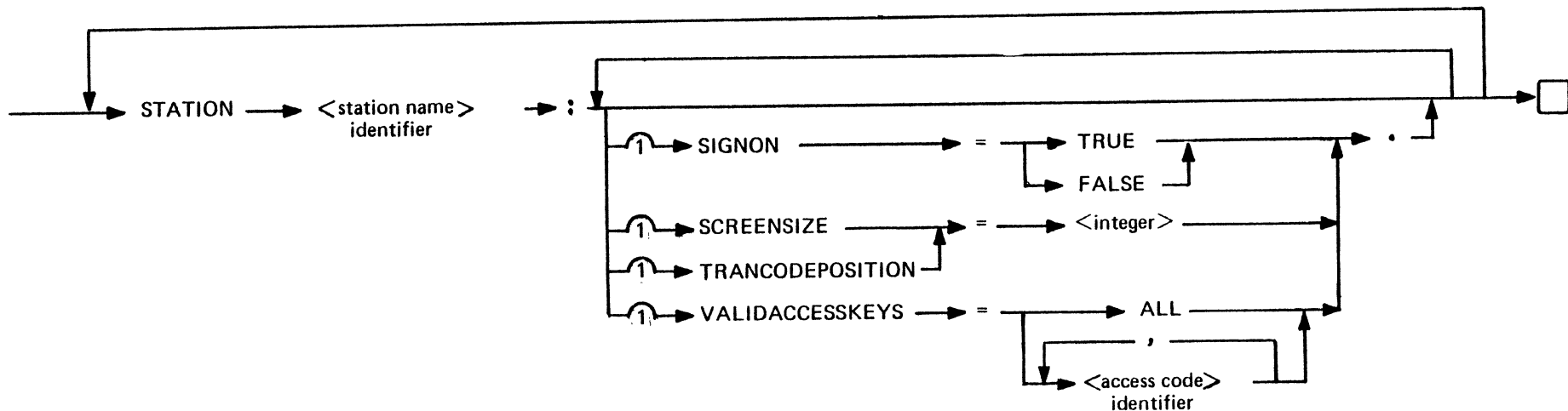


Figure A-6. Syntax of <STATION section>

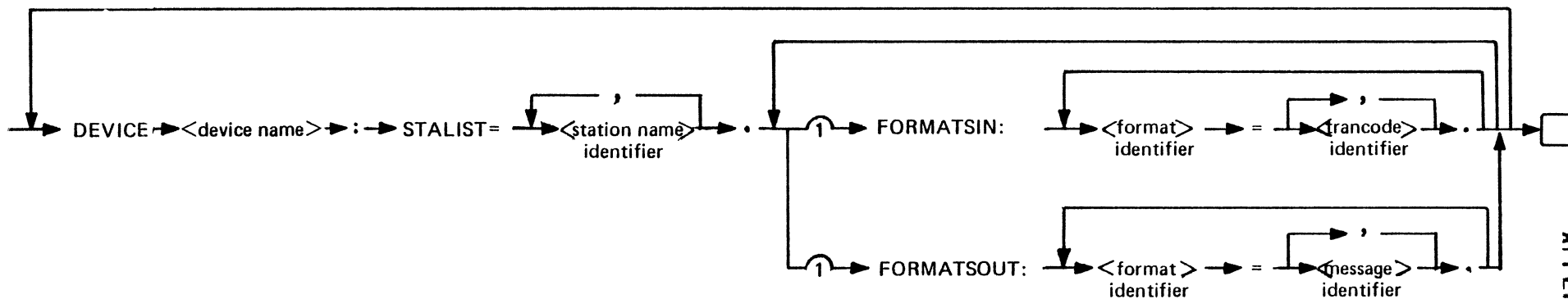


Figure A-7. Syntax of <DEVICE section>

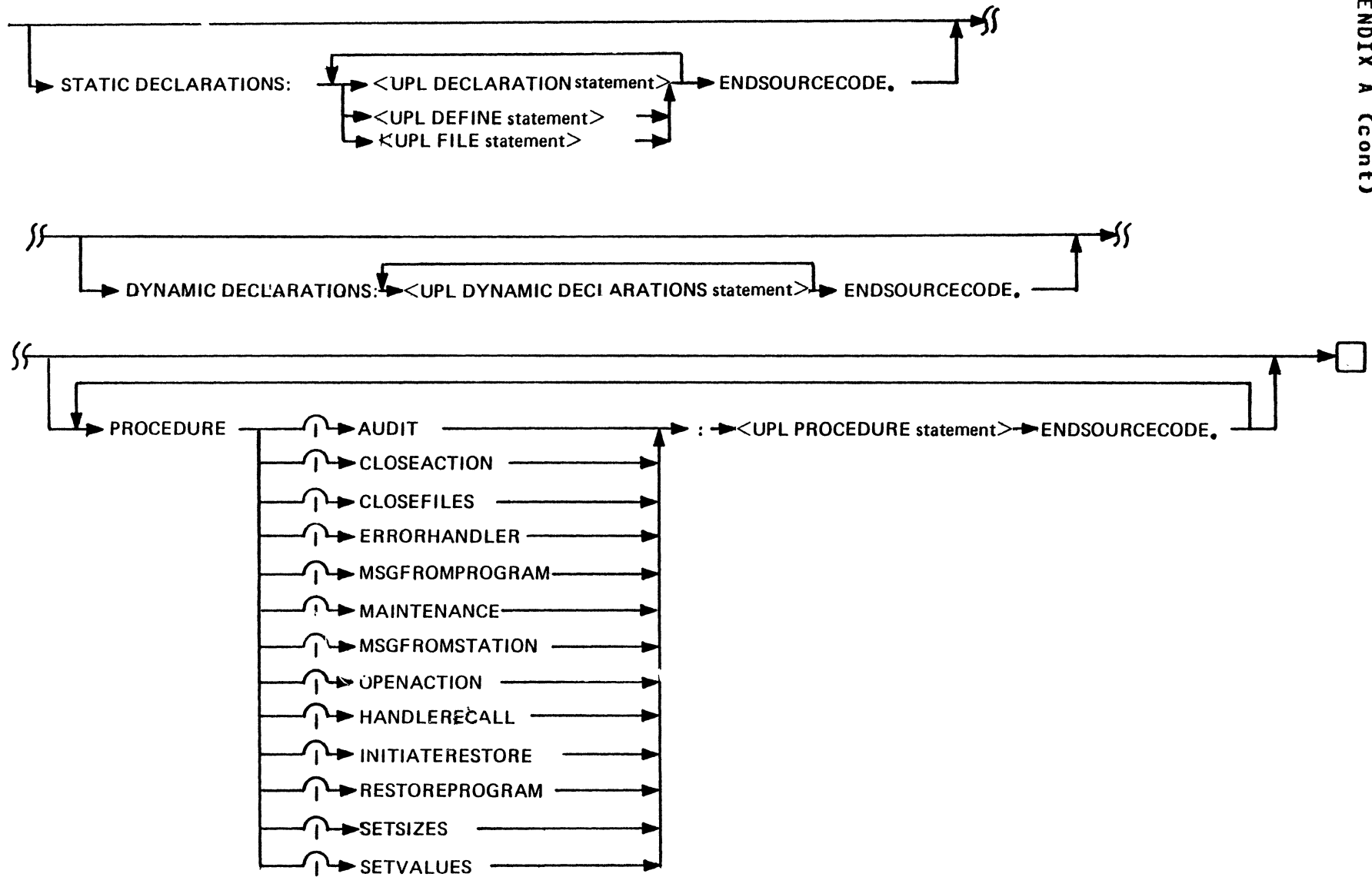


Figure A-8. Syntax of <MESS CODE section>

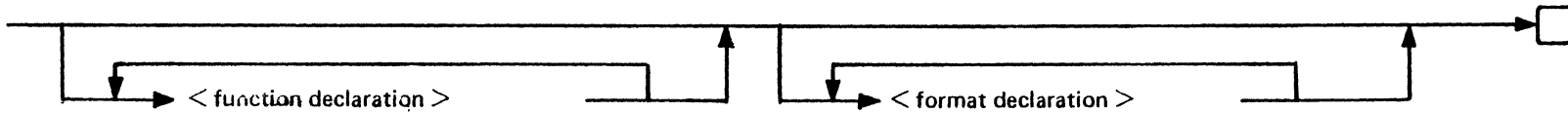


Figure A-9. Syntax of <FUNCTION and
FORMAT statement>

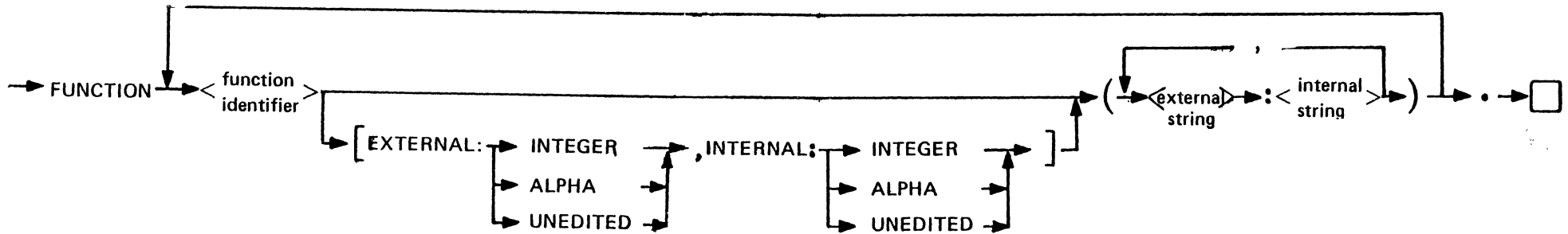


Figure A-10. Syntax of <function
declaration>

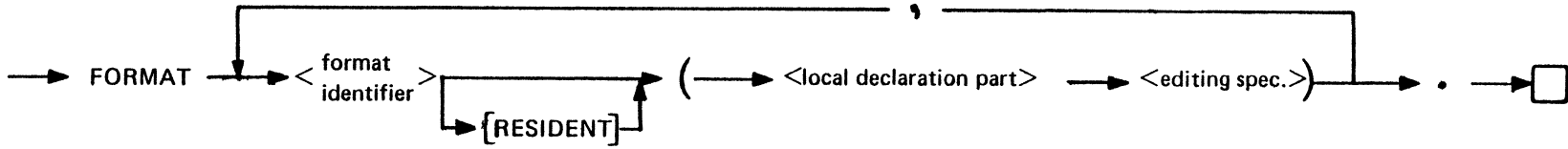


Figure A-11. Syntax of <format declaration>

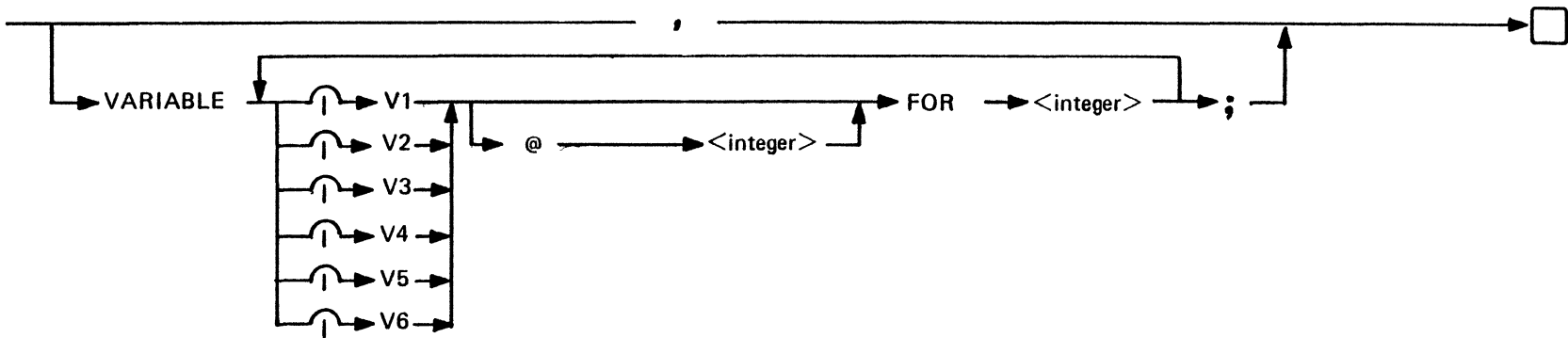


Figure A-12. Syntax of <local declaration part>

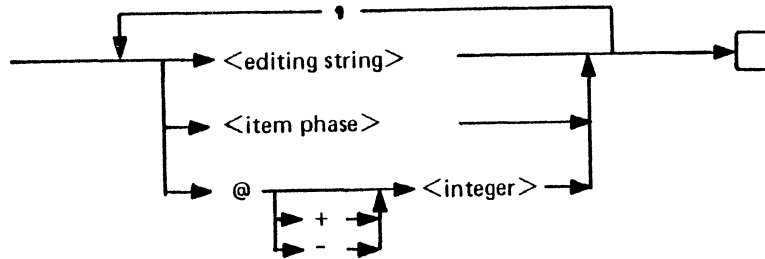


Figure A-13. Syntax of <editing specification>

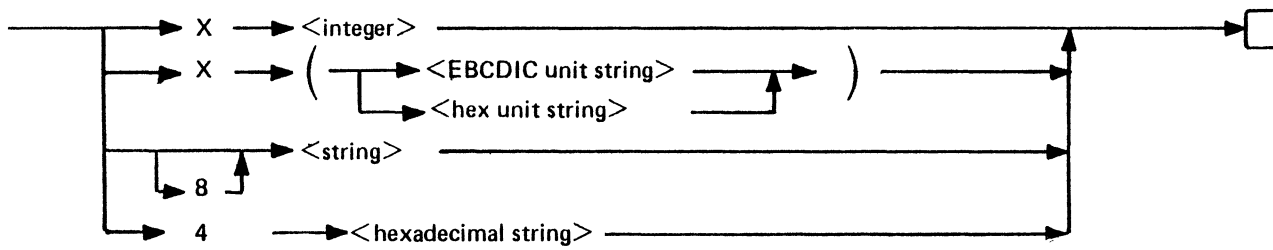


Figure A-14. Syntax of <editing string>

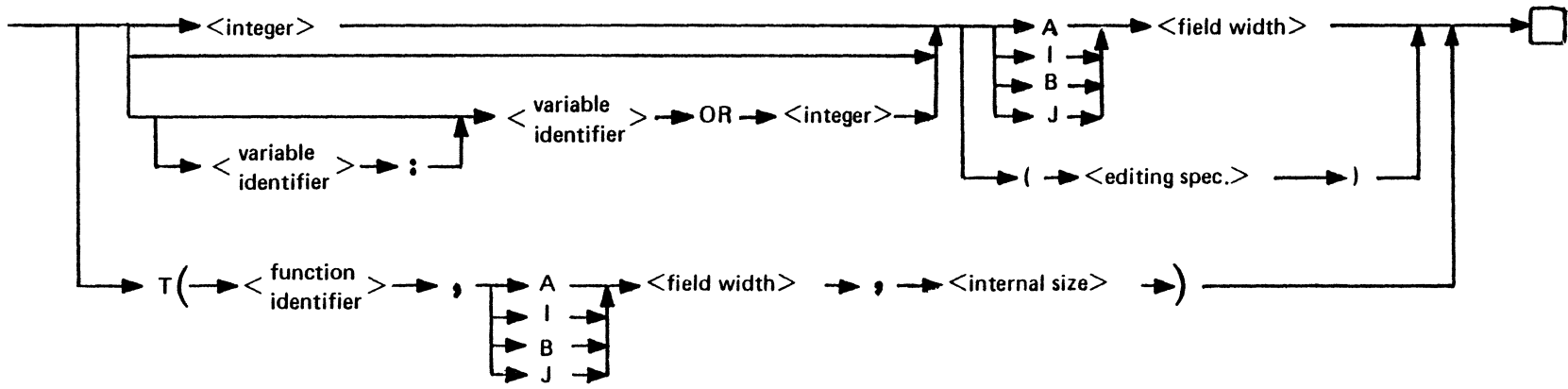


Figure A-15. Syntax of <item phrase>

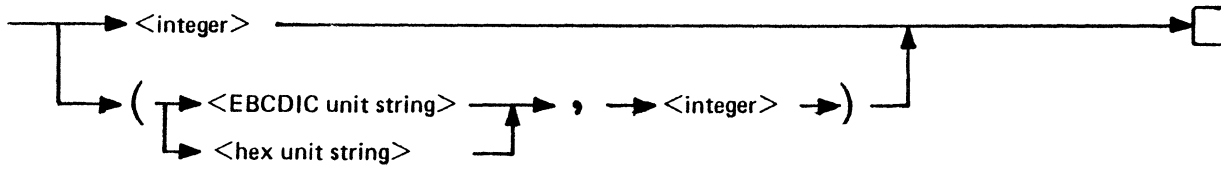


Figure A-16. Syntax of <field width>

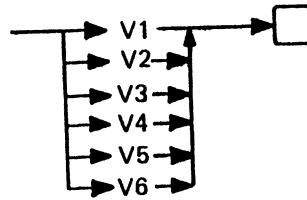


Figure A-17. Syntax of <variable identifier>



Figure A-18. Syntax of <internal size>

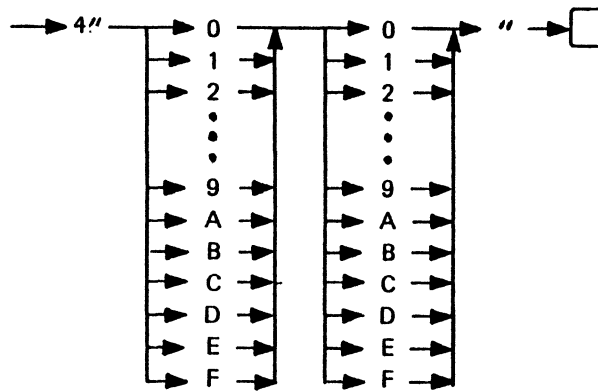


Figure A-19. Syntax of <hex unit string>

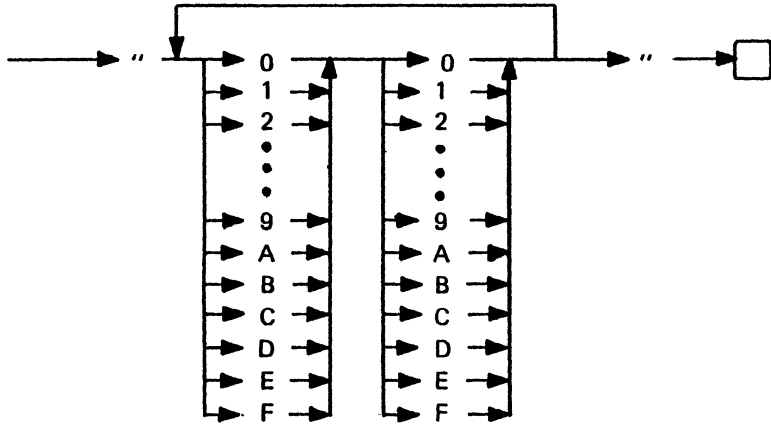


Figure A-20. Syntax of <hexadecimal string>

APPENDIX B
SUMMARY OF NETWORK CONTROL COMMANDS

GENERAL.

This appendix summarizes the network control commands. Refer to NETWORK CONTROL COMMANDS in section 3 for a definition of syntax conventions.

SECURITY CONTROL COMMANDS.

SIGN ON.

* SGN user-ID

SIGN OFF.

* BYE

ENABLE USER.

* EUS user-ID

DISABLE USER.

* DUS user-ID

PROGRAM CONTROL COMMANDS.

EXECUTE PROGRAM.

* EX program-name execute-option-list

HALT APPLICATION PROGRAM.

* HAP program-name

MCS CONTROL COMMANDS.

AUDIT OK.

* AOK

HALT SYSTEM.

* HLI KILL

APPENDIX B (cont)

MESSAGE CONTROL COMMANDS.

BROADCAST.

* BRC [station-specifier] ... : message-text

POP QUEUE.

* PQ station-specifier-1 ALL PRIN
SEND station-specifier-2

REPORT COMMANDS.

REPORT DATA DUMP.

* RDM PRINT

REPORT FILE STATUS.

* RES [file-name]

REPORT PROGRAM STATUS.

* RPS [program-specifier]

REPORT PROGRAM COUNTERS.

* RPC [program-specifier]

REPORT STATION STATUS.

* RSS [station-specifier]

REPORT STATION COUNTERS.

* RSC [station-specifier]

CHANGE COMMANDS.

CHANGE MONITOR FLAG.

* CME [flag-number] ... :
 D
 N

CHANGE STATION ADDRESS.

* CSA station-specifier I address
 Q

CHANGE STATION DIAGNOSTIC.

* CSQ station-specifier N
 Q

CHANGE STATION FREQUENCY.

* CSE station-specifier I frequency
 Q

CHANGE STATION MAX-RETRY

* CSM station-specifier retries

CHANGE STATION QUEUE.

* CSQ station-specifier-1 station-specifier-2

CHANGE STATION READY.

* CSR station-specifier R
 N

CHANGE STATION TRANSMISSION NUMBER.

* CSI station-specifier I transmission-number
 Q

AUDIT AND RECOVERY COMMANDS.

REFRESH.

- * REE

CLEAR DISABLED PROGRAM.

- * CLE program-specifier

RECOVER DATA BASE.

- * REC [data base specifier]

RESET BUSY STATUS

- * RBS station specifier

APPENDIX C
SYSTEM REQUIREMENTS

GENERAL.

Peripheral hardware needed to generate and execute a GEMCOS MCS includes:

- a. Card reader.
- b. Line printer.
- c. Console printer/keyboard.
- d. Disk subsystem. (If AUDIT is used, at least 4.6 megabytes are recommended, including MCP and Network Controller requirements but excluding user data files.)
- e. Single-line or multiline control.

A minimum of 24K bytes of main memory (exclusive of MCP requirements) is needed to generate an MCS. The smallest MCS that can be generated requires 7K bytes (exclusive of MCP and Network Controller requirements). Memory requirements increase for each option generated and for each additional entry in MCS tables.

The following chart can be used to estimate memory requirements in bytes for an MCS generated by the GEMCOS system. The calculation of the run structure as performed by the generator is a relatively complex process, and this chart is a simplification of it. Thus, the number calculated here is only an estimate of the memory requirements. It should be accurate to within 10 percent of the actual value.

The memory requirements for resident formats are difficult to estimate and it is assumed in the algorithm used in the chart that all formats are to reside on disk. To determine the amount of memory required for resident formats, compile the format descriptions with MCSTCL setting LIST in the <CONTROL statement>. The printer file labeled MCSRPT will be produced. Consult the pages which refer to FUN and FOR records adding the "lengths" of all functions and formats where "residence" is 1.

Note that the memory requirements of a B 1700 object code file can be obtained with the CODE/ANALYZER program found on a system release tape. However, in the case of a GEMCOS MCS, the calculation of the file space is not entirely correct since the MCS modifies several attributes of its files at run time.

FILE_SPACE.

MCSQUEUE

- Enter MAXTEXTSIZE on line 1. 1) _____
- Enter 50 on line 2. 2) _____
- If any program uses the PARTICIPATION interface enter 200 on line 3; otherwise enter 0. 3) _____
- Add lines 1, 2 and 3 giving line 4. 4) _____
- Enter the number of stations defined in the <STATION section> on line 5. 5) _____
- Enter 5 on line 6. 6) _____
- Multiply lines 5 and 6 giving line 7. 7) _____
- Enter QUEUEBUFFERS-1 on line 8. 8) _____
- Enter line 4 on line 9. 9) _____
- Multiply lines 8 and 9 giving line 10. 10) _____
- MCSTIC 12) 537

MCSPRINT

- If MONITORTRACE is TRUE, enter 248 on line 13. 13) _____

MCSAUDIT

- Enter AUDITRECORDSIZE on line 14. 14) _____
- Enter 357 on line 15. 15) _____
- If AUDIT is TRUE, add lines 14 and 15 giving line 16; otherwise, enter 0 on line 16. 16) _____

MCSOLDAUDIT

- If RECOVERY is required, enter line 15 on line 17; otherwise enter 0. 17) _____
- Add lines 11, 12, 13, 16 and 17 giving line 18. 18) _____

RUN STRUCTURE NUCLEUS

Enter 337 on line 19. 19) -----

F.I.B. dictionary

Enter 80 on line 20. 20) -----

GLOBAL VARIABLES

Enter 368 on line 21. 21) -----

If DATA DUMP is TRUE, enter 3 on line 22. 22) -----

If MESSAGERECALL or CHANGEREQUESTS is TRUE, enter 37 on line 23. 23) -----

If PROGRAMBOJEOJ is TRUE, enter 6 on line 24. 24) -----

If any program uses an interface of PARTICIPATION, enter 203 on line 25. 25) -----

If SYSTEMHALT is TRUE, enter 5 on line 26. 26) -----

If AUDIT is TRUE, enter 36 on line 27. 27) -----

If recovery is synchronized for any program, enter 234 on line 28; if RECOVERY = DATABASE or QUEUERESTORATION for any program, enter 11 on line 28. 28) -----

If the MCS is to format any message, enter 13 on line 29. 29) -----

If the MCS is to provide access control, enter 35 on line 30. 30) -----

If a CONTROLSTATION is specified, enter 5 on line 31. 31) -----

If any network control command is to be supported, enter 178 on line 32. 32) -----

Add lines 21 through 32 giving line 33. 33) -----

GLOBAL TABLES AND MESSAGE AREAS (Excluding formatting)

If any program uses an interface of PARTICIPATION, enter 203 on line 34. 34) -----

If any kind of recovery is required, enter AUDITRECORDSIZE + 30 on line 35; if only auditing is required, enter AUDITRECORDSIZE on line 35. 35) -----

APPENDIX C (cont)

Enter MAXTEXTSIZE on line 36.	36) _____
Enter 55 on line 37.	37) _____
Enter the number of stations defined in the station section on line 38.	38) _____
Enter 91 on line 39.	39) _____
Multiply lines 38 and 39 giving line 40.	40) _____
Enter the sum of all copies of all programs on line 41.	41) _____
Enter 51 on line 42.	42) _____
Multiply lines 41 and 42 giving line 43.	43) _____
Enter the number of trancodes defined in the PROGRAM section on line 44.	
Enter 13 on line 45.	45) _____
Multiply lines 44 and 45 giving line 46.	46) _____
Enter the number of programs defined in the PROGRAM section on line 47.	47) _____
Enter 9 on line 48.	48) _____
Multiply lines 47 and 48 giving line 49.	49) _____
Enter line 38 on line 50.	50) _____
Enter line 43 on line 51.	51) _____
Enter line 46 on line 52.	52) _____
Enter 49 on line 53.	53) _____
Add lines 50 through 53 giving line 54.	54) _____
Multiply lines 50 and 54 giving line 55.	55) _____
Enter 8 on line 56.	56) _____
If an <ACCESS CONTROL statement> is present, divide line 55 by line 56 giving line 57.	57) _____
Add lines 34, 35, 36, 37, 40, 43, 46, 49 and 57 giving line 58.	58) _____

GLOBAL FORMATTING TABLES AND MESSAGE AREAS

Enter line 4 on line 59.	59) _____
Enter number of functions defined on line 60.	60) _____
Enter 3 on line 61.	61) _____
Multiply lines 60 and 61 giving line 62.	62) _____
Enter the number of formats defined on line 63.	63) _____
Enter 3 on line 64.	64) _____
Multiply lines 63 and 64 giving line 65.	65) _____
Enter line 60 on line 66.	66) _____
Enter the number of message-IDs defined in the DEVICE section on line 67.	67) _____
Add lines 66 and 67 giving line 68.	68) _____
Enter the number of devices defined in the DEVICE section on line 69.	69) _____
Multiply lines 68 and 69 giving line 70.	70) _____
Enter 10 on line 71.	71) _____
Multiply lines 70 and 71 giving line 72.	72) _____
Enter 8 on line 73.	73) _____
Divide line 72 by line 73 giving line 74.	74) _____
Enter line 67 on line 75.	75) _____
Enter 6 on line 76.	76) _____
Multiply lines 75 and 76 giving 77.	77) _____
Enter 384 on line 78.	78) _____
Add lines 59, 62, 65, 74, 77 and 78 giving line 79.	79) _____

APPENDIX C (cont)

GLOBAL PROCEDURE VARIABLES

If MONITORTRACE is TRUE, enter 45 on line 80.	80) -----
Enter 145 on line 81.	81) -----
Enter 0 on line 82.	82) -----
Enter 18 on line 83.	83) -----
If stations have been assigned SCREENSIZE values in the STATION section, add lines 82 and 83 giving line 84.	84) -----
If DATADUMP is TRUE, enter 303 on line 85.	85) -----
Enter 0 on line 86.	86) -----
Enter 0 on line 87.	87) -----
If AUDIT is TRUE, add lines 86 and 87 giving line 88.	88) -----
Add lines 80, 81, 84, 85 and 88 giving line 89.	89) -----

LOCAL VARIABLES

Enter 41 on line 90.	90) -----
If any network control command is to be supported, enter 170 on line 91.	91) -----
If any message is to be formatted, enter 315 on line 92.	92) -----
Enter 123 on line 93.	93) -----
Enter largest value of lines 91, 92 and 133 on line 94.	94) -----
Add lines 90 and 94 giving line 95.	95) -----

MESS CODE

Enter the number assigned to VALUESTACKBITS on line 96.	96) -----
Enter 8 on line 97.	97) -----
Divide line 96 by line 97 giving line 98.	98) -----
Enter the number assigned to NAMESTACKENTRIES on line 99.	99) -----
Enter 3 on line 100.	100) -----
Multiply lines 99 and 100 giving line 101.	101) -----
Add lines 98 and 101 giving line 102.	102) -----
Add lines 19, 20, 33, 58, 79, 89, 95 and 102 giving line 103.	103) -----
Compute 10% of line 103 and enter on line 104.	104) -----
Add lines 103 and 104 giving line 105.	105) -----

OTHER MEMORY CONSIDERATIONS.

DICTIONARY CONTAINER	106)	<u>30</u>
MASTER CODE SEGMENT DICTIONARY	107)	<u>30</u>
MEMORY LINKS	109)	<u>100</u>
Add lines 106 through 109 giving line 110.	110)	<u>2150</u>
Add lines 18, 105, and 110 giving the Estimated Memory to Run:		-----

MCSTIC FILE DISK SPACE REQUIREMENTS.

A record in the MCSTIC file is 180 bytes long. The file normally contains from 40 to 100 records.

AUDIT FILE DISK SPACE REQUIREMENTS.

The disk space used for audit files depends greatly on the operational procedures used at a specific installation. The system must be able to hold at least two audit files on disk at one time since audit files can be stored on some off-line medium such as magnetic tape when closed.

A record in an audit file is 180 bytes by default. If AUDITRECORDSIZE is specified, the length of the record is determined by the user. The file contains AUDITPAGE SIZE*40 records.

APPENDIX D
MCS OUTPUT MESSAGES

GENERAL.

This appendix describes all network messages which the MCS generates to inform users of errors or other conditions. The general format of a system or data communications error message is:

****ERROR mmm nn : error-message**

The general format of an operator message is:

?? nn : message

The number nn is the message number. The number mmm identifies the procedure which discovers the error and is provided to aid in debugging.

If DATADUMP = TRUE were specified in TCL, all error messages numbered less than or equal to 30 are accompanied by a table dump. For other messages, a dump can be obtained by using the RDN Network Control Command.

0_INVALID_HALT_PHASE.

During system shutdown, the current phase of shutdown is different from the one expected.

SUGGESTED ACTION:

Orderly shutdown may not be possible, and the MCS may have to be discontinued.

1_INVALID_STATUS_REQUEST.

During initialization, the MCS was sending status requests to the Network Controller and the Network Controller responded with an error message stating that some request was not valid. As a result, the corresponding entry in the MCS tables is not initialized.

SUGGESTED ACTION:

Discontinue the MCS, and execute it again.

2_HARDWARE_EXCEPTION_ON_MCSTIC.

While reading from or writing to MCSTIC, the MCS detected a physical disk error (such as a parity error). The MCS suspends itself awaiting keyboard input from the system operator.

SUGGESTED ACTION:

If the operator responds to the ACCEPT, the MCS attempts to shut down either gracefully (if SYSTEMHOLT = TRUE was specified in TCL) or directly. Otherwise the user must DS or DP the MCS.

3 EOF ON MCSTIC READ OR WRITE.

The MCS attempted to read from or write to MCSTIC using an invalid record number. The MCS suspends itself awaiting a response from the operator to an ACCEPT. If the operator responds, the MCS attempts to terminate normally.

SUGGESTED ACTION:

Analyze the dump for possible software error. Check the NPR record in MCSTIC for invalid record pointers. A new MCSTIC may have to be created.

4 INVALID RECORD ID IN MCSTIC.

The MCS read a MCSTIC record, but the ID field at the beginning of the record was not the one expected. The MCS suspends itself awaiting a response from the operator to an ACCEPT. If the operator responds, the MCS attempts to terminate normally.

SUGGESTED ACTION:

Analyze the dump for possible software error. Check the NPR record in MCSTIC for invalid record pointers. A new MCSTIC may have to be created.

5 INVALID OUTPUT MESSAGE - VARIANT.

The send routine detected an error in the header of a message that some other procedure was trying to send.

The meanings of the different possible variants are described below:

- a. HARDWR. An attempt was made to send a message to a station which cannot receive messages because of its hardware type (such as a card reader).
- b. TYPE. The message type field in the header does not have a valid value.
- c. SIZE. The message text size field in the header is greater than the maximum text size as declared at generation.
- d. LSN. The Logical Station Number in the header is invalid.

SUGGESTED ACTION:

In all of the above cases, the message cannot be sent and is dropped. It can be recovered from the data dump. This dump should be analyzed for a possible software problem.

6 INVALID MESSAGE SOURCE ON SEND.

When attempting to send a response to a Network Control Command, the MCS discovered that the LSN in the header was invalid.

SUGGESTED ACTION:

The message cannot be sent out and is dropped. It can be recovered from the data dump.

8 CONVERSATION MISMATCH IN AUDIT FILE.

The MCS attempts to perform a recovery. The conversation information in the MCSTIC file does not match the audit records.

SUGGESTED ACTION:

Review the audit files and rerun the MCS program. If the error recurs, execute a data dump and monitor trace.

10 INVALID MESSAGE VARIANT.

The MCS has received a message from the Network Controller whose variant field is not zero.

SUGGESTED ACTION:

The message is discarded, but can be recovered from the dump. The Network Controller is suspect.

11 INVALID FILE OPEN - STATION MYUSE.

An Application Program attempted to open a remote file containing a station whose MYUSE is not consistent with the open type.

SUGGESTED ACTION:

The Application Program must be discontinued. Either its OPEN must be modified or the MYUSE of the station must be changed in NDL.

12 INVALID FILE CLOSE - VARIANT.

An Application Program tried to close a remote file which it did not previously open. The close request is ignored. If "variant" is INVALID PROG NUMBER, then the program does not have any remote files open; if FILE NOT OPEN, then it has at least one other remote file open.

SUGGESTED ACTION:

Scrutinize the Application Program for possible logic flaws.

15 GOOD RESULTS RETURNED BAD LSN - LSN.

The MCS received a good results reply message from the Network Controller, but was unable to recognize the station indicated by LSN.

SUGGESTED ACTION:

Check MCS and Network Controller interface logic.

16 UNEXPECTED GOOD RESULTS FROM LSN - LSN.

The MCS received a good results reply message from the Network Controller for a station that was not expecting one.

SUGGESTED ACTION:

Check MCS and Network Controller interface logic.

17 RECOVERY OPENED CURRENT AUDIT FILE.

During recovery, the MCS attempted to open the current audit file for message reprocessing.

SUGGESTED ACTION:

Rerun the MCS. If problem persists, obtain a data dump and monitor trace.

18 BAD AUDIT RECORD DURING RECOVERY.

During recovery, the MCS attempted to read or write an audit file with an invalid key.

SUGGESTED ACTION:

Rerun the MCS. If problem persists, obtain a data dump and monitor trace.

19 FATAL ERROR FROM RESTART PGM - PROGRAM.

This error immediately follows the error explaining the nature of the problem.

SUGGESTED ACTION:

Check the logic of the Restart Program as well as any Application Programs involved.

20 BAD FILE # IN DETACH MSG.

The MCS received a detach message from the Network Controller but was unable to associate the remote file number with any known program.

SUGGESTED ACTION:

Check the MCS and Network Controller interface logic and analyze the data dump (if present) to obtain the program name.

21 BAD PGM # IN DETACH MSG.

The MCS received a detach message from the Network Controller for a program that was not running.

SUGGESTED ACTION:

Check the MCS and Network Controller interface and corresponding data dump.

31 POSSIBLE FRACTURED FORMAT AT LSN <LOGICAL STATION NUMBER>.

Screen wraparound sent a message containing a format to the indicated station. This is a warning message indicating that the format could be split in the middle of a forms field.

32 OPEN DENIED - MAXCOPIES EXCEEDED.

An attempt made to execute more than the maximum number of copies of a given program.

SUGGESTED ACTION:

DS or DP the suspended program. Regenerate TCL if program is a User Program and more copies are desired.

33 CANNOT HAP UTIL PGM FROM SPO OR CRD.

An HAP Network Control Command cannot be entered from the console printer or the card reader for programs which have been described in TCL as a Utility Program.

SUGGESTED ACTION:

Enter the command at the station(s) which entered the EX command for this program.

34 OPEN DENIED - OPEN OUTPUT CONFLICT.

A program opened a remote file output only but the MCS detected one or more of the following conflicts:

- a. Program must be declared NONPARTICIPATION.
- b. Program must not be an MCS.
- c. Program must not be a User Program.

SUGGESTED ACTION:

Re-evaluate program requirements and either modify program remote file open or modify TCL.

36 OPEN DENIED - SYSTEM SHUT DOWN.

An Application Program attempted to open a remote file during system shutdown.

SUGGESTED ACTION:

The program must be discontinued.

37 OPEN DENIED ON REMOTE FILE.

An Application Program attempted to open a remote file to which no stations could be attached.

SUGGESTED ACTION:

Check the FAMILY statement in the FILE section of NDL and the STATION section of the TCL deck.

38 SYSTEM GOING DOWN.

This message is sent to any station which attempts to enter a message during system shutdown. The input message is ignored.

39 INVALID PROGRAM NUMBER.

The MCS has detected that a program has gone to EOJ but is not recognized by the MCS as an active program.

SUGGESTED ACTION:

Use the RDM PRINT Network Control Command and the DM console keyboard commands to obtain diagnostic information. A monitor listing may also be helpful.

40 PROGRAM NOT KNOWN.

A program not defined in TCL was referenced in an EX or IAP Network Control Command.

SUGGESTED ACTION:

Use the RDM PRINT command to obtain a list of valid names.

41 INVALID PROGRAM NAME PROGRAM-NAME.

The program named in the Network Control Command either does not exist or does not have a remote file open.

42 PROGRAM NOT IN DISK DIRECTORY PROGRAM-NAME.

The program name in the EX Network Control Command is not on disk.

43 INVALID STATION NAME STATION-NAME.

The station name in the Network Control Command was not recognized.

44 NO ACTIVE FILE NAMED FILE-NAME.

The file named in the Network Control Command either does not exist or has not been opened.

45 INVALID INPUT MESSAGE - VARIANT.

When reading from its input queue, the MCS detected an error in the input message. The type of error is identified by the variant:

- a. TYPE. The message type is not in the range 0 thru 4.
- b. SIZE. The text size is greater than that which the TCL parameter MAXTEXTSIZE allows.

In either case the message is ignored, but is printed for debugging purposes.

SUGGESTED ACTION:

A type error in a message from an Application Program means that the program has put a value other than 0 (zero) in the type field of the actual key. In case of a SIZE error, longer messages can be allowed by changing MAXTEXTSIZE and regenerating.

46 INVALID OPERATION MNEMONIC.

The Network Control Command mnemonic was not recognized.

47 INVALID OPTION ITEM.

Some item expected as an option in a Network Control Command is invalid.

48 INVALID CHANGE DATA ITEM.

The value of a data item in a Change Network Control Command is invalid.

49 COMMAND MISSING DATA ITEM.

An item is missing from a Network Control Command.

50 EX COMMAND IGNORED - STA ATTACHED.

An EX Network Control Command was entered from a station when the station was already attached to an Assignment or a Utility Program.

SUGGESTED ACTION:

If the station is attached to a Utility Program, enter an HAP for that program and retry the EX. If the station is attached to an Assignment Program, all EXs from this station will return an error until the Assignment Program closes its remote file.

51 EX COMMAND LOCKOUT - TRY AGAIN.

The EX command was temporarily not available; therefore, retry the command.

52 TRANCODE'S PGM NOT ONDEMAND.

The station entered a valid trancode, but the program is not currently executing and was not declared in TCL with the statement "EXECUTE = ONDEMAND".

SUGGESTED ACTION:

Send a message to the CONTROLSTATION (or system operator) requesting that the necessary program be executed or regenerated with EXECUTE = ONDEMAND.

53 INVALID MONITOR FLAG.

The flag specification in a CMF command is invalid.

54 INVALID STATION LIST.

A BRC command contains more stations in the destination list than exist in the system.

SUGGESTED ACTION:

Use fewer stations in the list. If the list is null, the message is sent to all stations.

55 FILE NOT OPEN.

The MCS received a file close from a program whose remote file is not open.

56 INVALID INPUT - MISSING SIGNAL CHAR.

The MCS received a message from the console keyboard and no signal character was present at the beginning of the message.

SUGGESTED ACTION:

Re-enter the command with the signal (as declared in TCL) in position 1.

57 TRANCODE'S PGM DIDN'T OPEN THIS STATION.

The remote file which was opened by the program which handles this trancode did not contain this station.

SUGGESTED ACTION:

Change the FAMILY statement for this file in NDL.

59 NO TRANCODE AND STA NOT ATTACHED.

The message entered did not have a trancode (or a message-ID if formatting is being used), and the station is not attached to a Utility or Assignment Program.

60 USER ID NOT ALLOWED AT THIS STATION.

This user is not allowed to sign-on at this station.

61 SECURITY FAILURE - INVALID USER-ID.

The user-ID in a SGN command is not recognized by the system.

62 SECURITY FAILURE - USER-ID DISABLED.

The user-ID in a SGN command is disabled.

SUGGESTED ACTION:

The user cannot sign on until the user-ID is enabled by the EUS command.

63 SIGN-ON NOT REQUIRED.

A SGN command was received from a station which either does not require signing on or is already signed on.

SUGGESTED ACTION:

Proceed with normal transactions as if signed on.

64 SECURITY FAILURE - SIGN-ON REQUIRED.

The MCS received a message (other than a SGN command) from a station which requires signing on but is currently signed off. The message just entered is ignored.

SUGGESTED ACTION:

Sign on with a SGN command.

65 SECURITY FAILURE - RESTRICTED PROGRAM.

A message was entered by a user who is not allowed to use the destination program. The input message is ignored.

66 SECURITY FAILURE - RESTRICTED TRANS.

A message was entered by a user who is not allowed to use the transaction code in that message. The message is ignored.

67 SECURITY FAILURE - RESTRICTED NCC.

A Network Control Command was entered from a station that is not allowed to execute that command. Most commands are allowed only from the CONTROL STATION.

68 SIGN-ON COMPLETE AT TIME ON DATE.

A SGN command was successfully processed.

SUGGESTED ACTION:

The user is now signed on, and interaction with the system may begin.

69 SIGN-OFF COMPLETE AT TIME ON DATE.

A BYE command was successfully processed.

SUGGESTED ACTION:

Interaction with the system from this station is prohibited until some user successfully signs on again.

70 CHANGE DENIED.

The Network Controller responded to a network-change request with a change-denied message.

SUGGESTED ACTION:

The Network Controller should be checked to determine the reason for denial.

71 INVALID CHANGE TYPE.

The Network Controller found the type code in a network-change request to be invalid.

SUGGESTED ACTION:

The maintenance module of the MCS should be checked for possible software error.

72 INVALID CHANGE RESULT.

The Network Controller made an invalid response to a change request from the MCS.

SUGGESTED ACTION:

There may be a problem in the logic of the Network Controller.

73 INVALID CHANGE COMMAND.

The Network Controller responded to a change request from the MCS, but the response is not what the MCS expected.

SUGGESTED ACTION:

Investigate the maintenance module of the MCS for a possible software error.

74 SEND DENIED ON CHANGE.

The MCS found an error in a change request sent from the MCS to the Network Controller.

SUGGESTED ACTION:

The change command cannot be processed. Network activity continues, but further change requests may be ignored. If more changes are critical, the system must be shut down and restarted.

75 STATION NOT RUNNING.

An attempt was made to HAP a Utility Program from a station. Either the station is not attached to any program or it is not attached to the specified program.

76. HAP COMMAND LOCKOUT, TRY AGAIN.

The HAP command was temporarily unavailable; therefore, retry the command.

77. OPEN DENIED - PROG ALREADY RUNNING.

An Assignment Program attempted to open a remote file when there already was a copy of that program running. Only user programs can have multiple copies running.

SUGGESTED ACTION:

The Assignment Program must be DSed.

78. CANNOT EX UTIL PGM FROM CARD OR SPD.

An EX Network Control Command cannot be entered from the Control station or card reader for programs which have been described in TCL as Utility Programs.

SUGGESTED ACTION:

Either enter the EX command from a station or make the regenerate run of TCL to describe this program as an Assignment Program or User Program.

79. OPEN DENIED - UTIL PGM NOT EXPECTED.

A program described in TCL as a Utility Program attempted to open a remote file, and the MCS was not expecting the open request (i.e., the program was not executed from a station).

SUGGESTED ACTION:

DS the program and execute it from the station that is to use it.

80. UNEXPECTED ATTACH REPLY RECEIVED.

The MCS received an unexpected ATTACH REPLY from the Network Controller. The message is ignored.

SUGGESTED ACTION:

Check MCS and Network Controller logic.

81. ATTACH REPLY MISMATCH.

The MCS received an ATTACH REPLY which attached the wrong station or the wrong file or both.

SUGGESTED ACTION:

Check MCS and Network Controller logic.

82. ATTACH REQUEST DENIED BY NC, TYPE = DENIAL REASON.

The last attach request sent by the MCS was denied by the Network Controller. The station will not be attached. The possible values of denial reason are described in the B 1700 Systems Network Definition Language Reference Manual, form 1073715.

SUGGESTED ACTION:

Check MCS and Network Controller logic.

83. UNEXPECTED DETACH REPLY RECEIVED.

The MCS received an unexpected DETACH REPLY from the Network Controller. The message is ignored.

SUGGESTED ACTION:

Check MCS and Network Controller logic.

84. DETACH REPLY MISMATCH.

The MCS received a DETACH REPLY which detached the wrong station or the wrong file or both.

SUGGESTED ACTION:

Check MCS and Network Controller logic.

85. STATION NOT ATTACHED, INPUT IGNORED.

If no participating programs are declared in TCL and a station is not attached to a program, any message sent from that station results in this error. Entering *HAP without a program name when not attached also creates this error condition.

86. FMT_ERR, DEST_PTR_OUT_OF_BOUNDS.

The formatted message is larger than MAXTEXTSIZE.

SUGGESTED ACTION:

Check description of format in TCL.

87. FMT_ERR, SOURCE_PTR_OUT_OF_BOUNDS.

The raw (unformatted message) is larger than MAXTEXTSIZE.

SUGGESTED ACTION:

Check description of format in TCL.

88. FMT_ERR, NON-DIGIT_IN_INTEGER_FIELD.

The formatting code found a non-numeric character or an embedded blank in an integer field.

SUGGESTED ACTION:

Check the format description in TCL and the Application Program logic.

89. FMT_ERR, MISSING_SKIP_DELIMITER.

The message from the Application Program was missing a skip delimiter.

SUGGESTED ACTION:

Check the format description in TCL and the Application Program logic.

90. FMT_ERR, VARIABLE_REPEAT_ON_INPUT.

The station message invoked a format containing a variable repeat.

SUGGESTED ACTION:

Change the format description in TCL to exclude any variable repeats on input.

91 FMT_ERR, MISSING DELIMITER.

The message from the Application Program was missing a delimiter.

SUGGESTED ACTION:

Check the format description in TCL and the Application Program logic.

92 FMT_ERR, INVALID TRANSLATE FIELD.

The formatter attempted to translate a portion of the message from the Application Program using a TRANSLATE function, but the text did not match any of the internal strings of that function.

SUGGESTED ACTION:

Check the format and function description in TCL and the Application Program logic.

93 OPEN_DENIED - WRONG TCL INTERFACE.

Either the Application Program opened a remote file "with headers" and the TCL <INTERFACE statement> for that program was not declared to be INTERFACE = MCS, or the INTERFACE was declared to be MCS, but the program did not open its remote file "with headers".

SUGGESTED ACTION:

Correct the TCL INTERFACE parameter to match the program and regenerate MCSTIC.

94 INPUT IGNORED DURING RECOVERY.

The MCS received a message which is bound for a program that is undergoing recovery. The input message is ignored.

SUGGESTED ACTION:

When recovery is complete, enter the message again. Messages may be entered for programs not being recovered.

95 HAP_NOT_ALLOWED - PGM OPENED OUTPUT.

An attempt was made by a station to *HAP a program declared to be "output only". This is not accepted since the MCS cannot send an EOF to an output-only remote file.

SUGGESTED ACTION:

The program must be able to finish without aid from the MCS or it must be DSed.

96 CAN'T SEND EOF TO OUTPUT ONLY PGM # <PROGRAM>.

During MCS shutdown phase, programs are sent an EOF indicator which requires them to terminate. However, output only programs cannot be sent this EOF notice.

SUGGESTED ACTION:

The program must be terminated manually.

97 OPEN DENIED - PROGRAM NOT KNOWN.

A program which was not defined in TCL attempted to open a remote file.

SUGGESTED ACTION:

DS or DP program. Modify TCL deck to include this program.

98 OPEN DENIED - PROGRAM DISABLED.

A program known to the MCS attempted to open a remote file but the program was marked disabled.

SUGGESTED ACTION:

DS or DP program. Clear the program (*CLE) from the console printer or Control station. If all programs belonging to the data base are marked OK, recovery is then initiated.

99 OPEN DENIED - PGM OPENED 2 RMTE FILES.

A program attempted to open more than one remote file concurrently. The MCS permits a program only one remote open at any given time.

SUGGESTED ACTION:

DS or DP program. Disallow program from concurrent remote file opens.

100 ABNORMAL CLOSE DONE ON RESTART PGM - <PROGRAM>.

The MCS received a remote file close message from the Network Controller, but the Restart Program was still active. (See errors 101 and 102.)

SUGGESTED ACTION:

Check the MCS and Restart Program interface logic.

101 ENTER "OK" TO RESTART RESTART PGM.102 ENTER "NO" TO KILL THE MCS.

These two error messages are displayed on the Control station or console printer when a serious problem is encountered with the Restart Program. A message immediately preceding these errors explains the nature of the problem.

SUGGESTED ACTION:

Either enter "OK" or "NO" from the console printer by means of the accept mechanism.

103 OPEN DENIED-RESTART PGM NOT EXPECTED.

A Restart Program opened a remote file, but the MCS was not expecting a remote file to be open at that time.

SUGGESTED ACTION:

DS or DP program. Check the MCS and Restart Program interface logic.

104 END POSSIBLE DUPLICATE MESSAGES.

This error message always occurs in conjunction with error 105. A station receives this message after all possible duplicate messages have been displayed at the station at the finish of a synchronized recovery.

APPENDIX D (cont)

105 BEGIN POSSIBLE DUPLICATE MESSAGES.

This error message and error 104 are displayed at the end of a synchronized recovery if there are any messages for a station that the MCS cannot be sure the station has already seen. These messages are displayed between message 105 and message 104.

106 BUSY.

An attempt was made to enter a transaction at a station before the output from the previous transaction was received. This only occurs if TRANSACTIONMODE = TRUE for that station. The entered transaction is ignored.

SUGGESTED ACTION:

Wait for output from previous transaction or, enter the RBS network control command to reset the busy status of the station.

107 MSG NOT FOR SYNC DATABASE PGM.

An attempt was made to send a message to a program that is not part of a synchronized data base (as declared in TCL).

SUGGESTED ACTION:

Either restrict messages from this station to synchronized data base programs only or set the TCL station statement TRANSACTIONMODE to FALSE.

108 BEGIN RECOVERY OF (DATABASE/PROGRAM) - <number>.

109 END RECOVERY OF (DATABASE/PROGRAM) - <number>.

These two messages are displayed on the control station or console printer at the start and finish of recovery whenever recovery is initiated. For data base and synchronized recovery, the word DATABASE appears, otherwise the word PROGRAM appears. Additionally, any station that attempts to send a message to a program currently undergoing recovery receives message 109 upon completion of recovery.

110 GEMCDS ARCHIVAL RECOVERY.

111 ENTER ARCHIVAL SPECIFICATIONS.

These messages are displayed on the console printer whenever the MCS is executed in Archival Recovery mode.

SUGGESTED ACTION:

Enter the desired specifications as declared in this manual under ARCHIVAL RECOVERY.

112 INVALID ARCHIVAL SPECIFICATIONS.

The archival specifications entered at the console printer were incomplete or totally incomprehensible.

SUGGESTED ACTION:

Consult the ARCHIVAL RECOVERY section of this manual to determine the proper syntax.

113 MISSING OR UNRECOGNIZED WORD - <WORD>.

The archival specifications entered at the console printer contained a word (indicated in the error message) that was unrecognized or unexpected.

SUGGESTED ACTION:

Consult the ARCHIVAL RECOVERY section of this manual to determine the proper syntax.

114 "ALL" MUST BE ONLY NAME IN PGM LIST.

The archival specifications entered at the console printer contained program names in addition to the word "ALL".

SUGGESTED ACTION*

Either enter a list of program names or the word "ALL" at this point in the archival specifications.

115 BEGIN ARCHIVAL RECOVERY.**116 END ARCHIVAL RECOVERY.**

These two messages are displayed on the console printer at the start and finish of archival recovery, respectively. After displaying message 116, the MCS terminates.

117 NO INPUT ALLOWED: ARCHIVAL RECOVERY.

An attempt was made by a station to send a message to a program while the MCS was in ARCHIVAL RECOVERY mode. This and any other such request is ignored.

SUGGESTED ACTION:

Do not send any messages to programs during ARCHIVAL RECOVERY.

118 PROGRAM NOT IN DATABASE - <number>.

The archival specifications entered at the console printer contain a program name that does not belong to the previously mentioned data base.

SUGGESTED ACTION:

Check the archival specifications against the TCL specifications describing which programs belong to the data base.

120 MCSTIC TIME MISMATCH - <audit file-name>.

The MCSTIC date/time stamp in the MCSTIC file for the current audit file does not match the MCSTIC date/time stamp in the audit file.

SUGGESTED ACTION:

Check the validity of the MCSTIC file and the audit file named <AUDIT FILE-NAME>.

121 AUDIT FILE TIME MISMATCH - <audit file-name>.

The audit file date/time stamp in the MCSTIC file for the current audit file does not match the audit file date/time stamp in the audit file.

SUGGESTED ACTION:

Check the validity of the MCSTIC file and the audit file named <AUDIT FILE-NAME>.

122 CURRENT TIME LESS THAN LAST AUDIT.

The value of the current time maintained by the system clock is less than the time of the last audit performed by the MCS.

SUGGESTED ACTION:

Check the validity of the system time clock.

125 MCS LOST REPLY TO <NN> INPUTS TO PGM - <number>.

During synchronized recovery, the MCS detected <NN> input messages from a given station to a program for which no output messages were ever received.

SUGGESTED ACTION:

An inquiry into the data base should be made to see if the transactions in question are on the data base.

126 UNEXPECTED CLOSE FROM PGM - <number>.

A program that is part of a data base using audit and recovery was terminated abnormally. The data base is marked as needing recovery. Error message 127 always follows this message.

127 TO INITIATE RECOVERY, CLEAR PGM - <number>.

Either this program just terminated abnormally or an attempt was made to initiate recovery (by the MCS or the user). In either case, the program must be cleared before recovery can begin.

SUGGESTED ACTION:

Clear the disabled program using the *CLE command. If recovery does not begin, then at least one other program in the data base is still disabled and must also be cleared.

128 FILE MISSING - <audit file-name>.

During recovery, the named audit file was sought on disk by the MCS but was not found.

SUGGESTED ACTION:

The named audit file must be loaded on disk, and an *AOK command must be entered on the console printer.

129 FILE LOCKED - <audit file-name>.

During recovery, the MCS tried to open the named audit file but found it was locked by another program.

SUGGESTED ACTION:

The program that locked the named audit file must free it for use by the MCS, and an *AOK command must be entered on the console printer.

130 INCORRECT FILE - <audit file-number>.

During recovery, the MCS opened the named audit file and determined that one or both of the date/time stamps in the file did not match the value stored in the MCSTIC file.

SUGGESTED ACTION:

Check the named file to insure that it is the correct file.

131 WRONG AUDIT RECORD REQUESTED - PGM <number>.

During recovery, the MCS found an error in the audit file during an attempt to service a program. The program is DSed if it is part of a data base; otherwise, the MCS is terminated.

SUGGESTED ACTION:

Either bring the MCS back up or clear the DSed program.

132 UNEXPECTED TYPE 22 MSG FROM PGM - <number>.

This program sent the MCS an unsolicited type-22 message. A program can only send this message upon receipt of a type-21 message. The program is DSed.

SUGGESTED ACTION:

Investigate the logic of this program. Clear the program to initiate recovery.

133 NO AUDITED MESSAGES FOR REF NCC.

An *REF command was entered to recall the last audited output message for a station, but the MCS had no audited messages for this station.

134 OLD AUDIT FILE MISSING FOR REF NCC.

An *REF command was entered, but the last audited message for that station was in an audit file that is not on disk.

SUGGESTED ACTION:

Load the missing audit file on disk.

135 MISSING PROGRAM NAME.

A Network Control Command was entered that required either a program name or number, but none was found.

SUGGESTED ACTION:

Refer to syntax of the Network Control Command that was in error.

136 INVALID PROGRAM - <token>.

The program name or number entered (token) was found to be either invalid or unrecognized.

SUGGESTED ACTION:

Refer to the TCL specifications for a list of all valid program names.

137 PROGRAM NOT DISABLED - <program name or number>.

An attempt was made to clear a program (using the *CLE command) that was not disabled.

SUGGESTED ACTION:

If there is any doubt of the status of the program, recovery can be performed on that data base.

APPENDIX D (cont)

138 INVALID DATABASE NAME - <database name or number>.

The data base name or number entered was found to be either invalid or unrecognized.

SUGGESTED ACTION:

Refer to the TCL specifications for a list of all valid data base names.

139 PROGRAM DISABLED - <program name or number>.

An attempt was made to execute this program either by the *EX command or by entering a trancode for this program, but the program was disabled.

SUGGESTED ACTION:

Clear the program with the *CLE command and, thus, initiate recovery of this data base.

140 EOF ALREADY SENT TO - <program name>.

An attempt was made to send a *HAP command to a program more than once.

SUGGESTED ACTION:

If the program does not go to EOJ within a few minutes, investigate the EOF logic of the remote file of the program.

141 INPUT IGNORED; PROGRAM TERMINATING.

An attempt was made to send a message to a program that is in the process of halting (a *HAP command was performed on this program).

SUGGESTED ACTION:

Re-execute the program.

142 UNEXPECTED TYPE 25 MSG FROM PGM - <program number>.

A program sent the MCS a message with MCS-TYPE field of the Common-area header set to 25 without first receiving a type-24 message.

SUGGESTED ACTION:

Investigate the program's MCS interface logic.

143 TYPE 17 OR 18 MSG FROM PROGRAM - <program number>.

A program that was not declared in TCL to be a Restart Program sent a message to the MCS with MCS-TYPE field of the Common-area header set to 17 or 18.

SUGGESTED ACTION:

Investigate the program's MCS interface logic. Only Restart Programs can send type-17 or-18 messages.

144 REF NCC CANNOT COME FROM SPO.

An attempt was made to enter a *REF command from the console printer. This is not allowed and is ignored.

145 RESTART_PGM_RETURNED_BAD_DATABASE - <database-name>.

The MCS was expecting the Restart Program to return the name of the data base to be recovered, but the name returned did not match the MCS's expectations.

SUGGESTED ACTION:

Investigate the Restart Program's MCS interface logic. The <database-name> is what was received by the MCS.

146 RESTART_PGM_FOUND_ERROR - DATABASE <database-name>.

The Restart Program found an error while accessing the data base. See error messages 101 and 102 for further explanation and suggested action.

147 RESTART_PGM_RETURNED_BAD_PDT # - <PDT number>.

The Restart Program returned a bad value for the PDT number. This value was originally stored in the restart data set by the program that created the record.

SUGGESTED ACTION:

Investigate the program logic of all programs in the data base that deals with the type-23 message.

148 RESTART_PGM_RETURNED_BAD_PROG # - <PROG number>.

The Restart Program returned a bad value for the PROG number. This value was originally stored in the restart data set by the program that created the record.

SUGGESTED ACTION:

Investigate the program logic that deals with the type-23 message in all programs in the data base.

APPENDIX E
SUMMARY OF FILES

GENERAL.

The GEMCDS system files are summarized in the following table.

Table E-1

Summary of GEMCDS
System Files

Program	Internal Name	Input (I) Output (O)	Device	External Name
MCSSRC/OBJECT	MCSTIC	I/O	Disk	
	MCSFORMATS	I/O	Disk	
	MCSQUEUE	I/O	Remote	
	MCSBCKLG	I/O	Queue	
	MCSAUDIT	I/O	Disk	MCSAUDIT/AUDITnnn
	MCSOLDAUDIT	I/O	Disk	MCSAUDIT/AUDITnnn
	MCSPRINT	O	Printer	
	MCSTANK	I/O	Disk	MCSTANK/<random number>
MCSSIM	PRINT.OUT	O	Printer	MCSSIMPRT
	CARDS	I	Card	MCSSIMCRD
	MCSQUEUE	O	Queue	
MCSFIX	SOURCE	I	Disk	
	NEWSOURCE	O	Disk	
	CARDS	I	Card	
	LINE	O	Printer	
MCSGO		I	Disk	MCSGTS
		I	Disk	MCSFTS
		I	Card	MCSCRD
		O	Disk	MCSSRC
		O	Disk	MCSFIT
		I/O	Disk	MCSTMP
		O	Printer	MCSERN
MCSTCL	MCSIN	I	Card	MCSIN
	MCSLST	O	Printer	MCSLST
	MCSPRM	O	Disk	MCSTMP
	MCSTIC	O	Disk	MCSTIC
	MCSFORMATS	I/O	Disk	MCSFORMATS
	MCSTMP	I/O	Disk	MCSTMP1
	OLDMCSTIC	I	Disk	MCSTIC
	MCSRPT	O	Printer	MCSRPT
	DIRECTORY	I/O	Disk work	MCSDIRECTY
MCSRECALL	MCSTIC	I	Disk	MCSTIC
	MCSREM	I/O	Remote	MCSREM

Table E-1 (cont)
Summary of GEMCDS
System Files

Program	Internal Name	Input (I) Output (O)	Device	External Name
	MCSAUDIT MCPRT	I O	Disk Printer	MCSAUDIT/AUDITnnn MCPRT

APPENDIX F

COBOL74

GENERAL.

The 4.10 release of GEMCOS provides interface capabilities between the GEMCOS-generated MCS and COBOL74 programs. Burroughs COBOL74 complies to the coding standards established by the American National Standards Institute (ANSI) of 1974. (For further information about the COBOL74 language, refer to the B 1000 System COBOL74 Reference Manual, form number 1108883, for the 9.0 release of the B 1800/B 1700 systems.)

TCL REQUIREMENTS.

COBOL74 programs that interface with GEMCOS do not affect or require changes in the specifications established in the Transaction Control Language (TCL). However, the programs must be declared in the PROGRAM section of the TCL and defined as participating programs in order to facilitate COBOL74 interface.

NETWORK DEFINITION LANGUAGE (NDL) REQUIREMENTS.

The NDL/LIBRARY file, provided for the 9.0 release of the B 1800/B 1700 system, includes COBOL74 declarations and the COBOL74SEL request set. Both the declarations and the request set must be copied from the NDL/LIBRARY file and merged into the appropriate area in the user's NDL source file. POLL/SELECT devices, interfacing with COBOL74 programs, require a DIAGNOSTIC statement in the TERMINAL section of the network definition language. This statement must specify the request set, POLLTCTD, in the RECEIVE portion, and COBOL74SEL in the TRANSMIT portion of the statement. An example follows:

Example:

```
TERMINAL TD830:  
  ADDRESS = 2.  
  TRANSMISSION = 0.  
  REQUEST = CANDEPOLTD : RECEIVE, CANDESELTD : TRANSMIT.  
  DIAGNOSTIC = POLLTCTD : RECEIVE, COBOL74SEL : TRANSMIT.  
  BUFFERSIZE = 2000.  
  TYPE = 46.
```

The network controller is recompiled upon entering specifications such as those presented in the example above as well as merging declarations and the COBOL74SEL request set. (For additional information, refer to the B 1800/B 1700 Systems Network Definition Language (NDL) Reference Manual, form number 1073715.)

COBOL74 PROGRAM REQUIREMENTS.

Before COBOL74 programs send messages to the GEMCOS MCS, they must execute the ENABLE <cd-name> or the RECEIVE INPUT <cd-name> command. Either command generates a network-control FILE OPEN message (TYPE = 18), which is sent to the MCS. The MCS accepts no messages from a COBOL74 program until it receives the FILE OPEN message.

RESTRICTIONS.

Multiple COBOL74 programs must not open the same remote file. If this occurs, the results are unpredictable due to the 9.0 system software implementation. This restriction limits the use of the MAXCOPIES statement.

APPENDIX G

TCL SIZE LIMITATIONS

The TCL compiler has several size limitations. They restrict the maximum number of various entities such as stations or programs which can be declared in TCL. These size limitations follow:

The <Maxcopies statement> must be less than	256
The number of programs must be less than	217
The number of trancodes must be less than	606
The number of message IDs must be less than	1023
The number of stations must be less than	131
The number of devices must be less than	1023
The number of functions must be less than	1023
The number of formats must be less than	1023
The number of users must be less than	1023
The number of trancodes + the number of stations + the number of programs must be less than	1367
The number of programs * (83 + <sum of maxcopies>) must be less than	65535
The number of devices * the number of trancodes * 10 must be less than	65535
The number of stations * (48 + the number of trancodes + the number of programs) must be less than	65535
The number of message-IDs * number of devices * 10 must be less than	65535

INDEX

Access control, 2-3
ACCESS CONTROL statement, 3-73
Application-program interface, 2-1
Archival recovery, 9-13, 9-14
ATTACH MESSAGE statement, 3-100
Audit, 2-6
AUDIT ALL OUTPUT statement, 3-23
AUDIT ASSIGNMENT statement, 3-95
AUDIT FILE PACK ID statement, 3-38
AUDIT OK (AOK) command, 3-141
AUDIT OUTPUT statement, 3-96
AUDIT PAGE SIZE statement, 3-37
AUDIT RECORD SIZE statement, 3-36
Audit & Recovery commands, 3-159 thru 3-161
AUDIT TRANSACTIONS statement, 3-94
Auxiliary programs, section 8

Basic components, 3-5 thru 3-10
Basic symbols, 3-4
BROADCAST (BRC) command, 3-143

Change commands, 3-151 thru 3-158
CHANGE MONITOR FLAG (CMF) command, 3-151
CHANGE REQUESTS statement, 3-21
CHANGE STATION ADDRESS (CSA) command, 3-152
CHANGE STATION DIAGNOSTIC (CSD) command, 3-153
CHANGE STATION FREQUENCY (CSF) command, 3-154
CHANGE STATION MAXIMUM RETRY (CSM) command, 3-155
CHANGE STATION QUEUE (CSQ) command, 3-156
CHANGE STATION READY (CSR) command, 3-157
CHANGE STATION TRANSMISSION (CST) command, 3-158
Character set, 3-3
CHECKPOINT INTERVAL statement, 3-39
CLEAR BUSY FLAG (CBF) command, 3-139
CLEAR DISABLED PROGRAM (CLE) command, 3-160
Close files, 3-123
Common-area header, 2-1, 2-2, 2-6, 3-101, 3-126 thru 3-130
COMMON SIZE statement, 3-89
COMPILE OPTIONS statement, 3-29
Compiling with MCSTCL, 7-1
Console or card reader input, 7-1
CONTINUOUS LOG-ON statement, 3-109
CONTROL statement, 3-13 thru 3-17
CONTROL STATIONS statement, 3-71
Controlled shutdown, 2-6
Conventions for a "RECOVERY = SYNCHRONIZED" specification, 9-9, 9-10

INDEX (cont)

DATA BASE NAME statement, 3-93
Data base recovery (nonsynchronized), 9-7
DATA DUMP statement, 3-22
Debugging Aids, 2-8
Deck Description, 3-11, 3-12
DEFINITION section, 3-63 thru 3-74
DETACH MESSAGE statement, 3-101
DEVICE section, 3-110 thru 3-114
DISABLE USER (DUS) command, 3-137
Dynamic declarations, 3-117

ENABLE USER (EUS) command, 3-136
Error handler, 3-123
Error handling, 2-5
EXECUTE PROGRAM command, 3-136
EXECUTE statement, 3-90, 3-91
Executing an MCS, 7-1

FAMILY statement of NDL, 2-1, 2-2
Files, GEMCOS system, appendix E
FORMAT AND FUNCTION statement list, 3-46, 3-47
Format declaration, 3-51 thru 3-57
Formatting, 2-9
Formatting errors, 3-58 thru 3-69
Function declaration, 3-48 thru 3-50

GEMCOS editing phrases, 6-1, 6-2
Generation, 1-1
GLOBAL section, 3-19 thru 3-71

HALT APPLICATION PROGRAM (HAP) command, 3-140
HALT SYSTEM (HLT) command, 3-142
Handle recall, 3-124

Identifiers, 3-4
INPUT FORMATS statement, 3-113
INTERFACE statement, 3-80

Logical Station Number (LSN), 2-1

MAXIMUM COPIES statement, 3-98
MAXIMUM TEXT SIZE statement, 3-40
MCS control commands, 3-141, 3-142
MCS interface, 3-82 thru 3-84
MCS output messages, appendix d
MCS program, 1-1, 1-2
MCSFIX program, 1-3, 8-3 thru 8-5
MCSIN, 3-1
MCSRECALL, 1-3
MCSSIM program, 8-1, 8-2
MCSTCL (MCS compiler), 1-3, 3-1
MCSTIC (Table Information Control file), 1-3
MCSTIC FILE NAME statement, 3-18
Mergeable external source statements, 2-7

INDEX (cont)

MESSAGE BROADCAST statement, 3-23
Message control commands, 3-143, 3-144
MESSAGE RECALL statement, 3-24
Message recovery, 2-6
Message routing, section 4
MESS CODE section, 3-115 thru 3-119
MESS procedures, 3-120 thru 3-125
Messages, network, appendix D
Message type, 2-1
Metalinguistic formulas, 3-2
Monitor, 2-9, 3-131
MONITOR TRACE statement, 3-26
MONITOR TRACE ON statement, 3-45

NAME-STACK ENTRIES statement, 3-31
NCC OK RESPONSE, 3-34
NDL, 2-1, 2-2
Network administration, 2-4, 2-5
Network Controller (NC), 2-4 thru 2-8
Network Control Commands (NCCs), 2-4 thru 2-8, 3-132 thru 3-151
Network restoration, 2-5
NONPARTICIPATION interface, 3-80, 3-81
Nonsynchronized and synchronized data base recovery, 9-3
No recovery, 9-1, 9-2

OBJECT-CODE FILE NAME statement, 3-30
OPEN MESSAGE statement, 3-99
OUTPUT FORMATS statement, 3-114

PARTICIPATION interface, 3-82 thru 3-84
POP QUEUE (PQ) command, 3-144
Procedure define list, 3-118, 3-119
Program abort, 9-6
PROGRAM BOJ EOJ statement, 3-25
Program control commands, 3-138 thru 3-140
PROGRAM section, 3-75 thru 3-101
PROGRAM TITLE statement, 3-87

QUEUE BUFFERS statement, 3-42
QUEUE DEPTH statement, 3-41
QUEUE NAME statement, 3-43
Queue restoration recovery, 9-2, 9-3

RECALL PROGRAM statement, 3-70
RECOVER DATA BASE (REC) command, 3-151
Recovery, section 9
Recovery after system failure, 9-7
Recovery cycle, 9-11, 9-12
Recovery procedure, 7-2
Recovery processing, 9-7
Recovery-related conventions, 9-9
RECOVERY statement, 3-92
REFRESH (REF) command, 3-159
Remote file, 2-1, 2-2, 9-4 thru 9-6

INDEX (cont)

Remote program execution, 2-9
Report commands, 3-145 thru 3-150
REPORT DATA DUMP (RDM) command, 3-145
REPORT FILE STATUS (RFS) command, 3-146
REPORT PROGRAM COUNTERS (RPC) command, 3-148
REPORT PROGRAM STATUS (RPS) command, 3-147
REPORT STATION COUNTERS (RSC) command, 3-149
REPORT STATION STATUS (RSS) command, 3-150
RESIDENCE statement, 3-88
Restart program, 9-11
RESTART PROGRAM statement, 3-97
Routing, 2-4, 2-8
 (See also Transaction-Based Routing)

Security, 2-2, 2-3, 2-4
Security control commands, 3-134 thru 3-137
Screen wraparound, 2-8, 2-9
SCREEN SIZE statement, 3-105
SIGNAL CHARACTER statement, 3-35
SIGN-ON statement, 3-104
SIMULATION statement, 3-44
SOURCE-CODE FILE NAME statement, 3-31
Static declarations, 3-116
STATION LIST statement, 3-112
STATION section, 3-102, 3-103
STATION section of NDL, 2-1
STATUS REPORTS statement, 3-27
Strings, 3-9
Summary of files, appendix E
Summary of Network Control Commands, appendix B
Summary of TCL syntax, appendix A
Supervisory MCS, 2-9
Synchronized recovery, 9-8
Syntax conventions, 3-1
SYSTEM HALT statement, 3-28
System operation, section 7
System requirements, appendix C
System structure, 1-3

TRANCODE statement, 3-85
Transaction-Based Routing (TBR), 2-2, 2-3
TRANSACTION CODE POSITION statement, 3-106
Transaction Control Language (TCL), 1-3, section 3
Transaction Control Language compiler, 1-3
TRANSACTION MODE statement, 3-108
Transaction processing, 9-6
Types of recovery, 9-2 thru 9-14

INDEX (cont)

User Programming Language (UPL), 1-2
User programs, 3-78, 3-79
Utility programs, 3-77

VALID ACCESS KEYS statement, 3-107
VALUE-STACK BITS statement, 3-33