Burroughs

*Reference Manual*

# B 20 Systems

# Burroughs Multipoint Communications Service (BMULTI)

*(Relative to release level 5.0)*

*Distribution Code SA*

*Priced Item*
*Printed in U.S.A.*
*April 1985*

*1182284*

Burroughs

*Reference Manual*

# B 20 Systems Burroughs Multipoint Communications Service (BMULTI)

*(Relative to release level 5.0)*



*Distribution Code SA*

*Priced Item*
*Printed in U.S.A.*
*April 1985*

*1182284*

Burroughs cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued from time to time to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded, using the Documentation Evaluation Form at the back of the manual, or remarks may be addressed directly to Burroughs Corporation, Corporate Product Information East, 209 W. Lancaster Ave., Paoli, PA 19301, U.S.A.

## LIST OF EFFECTIVE PAGES

| Page | Issue |
|---|---|
| iii | Original |
| iv | Blank |
| v thru viii | Original |
| 1-1 and 1-2 | Original |
| 2-1 thru 2-11 | Original |
| 2-12 | Blank |
| 3-1 thru 3-47 | Original |
| 3-48 | Blank |
| 4-1 thru 4-14 | Original |
| A-1 thru A-28 | Original |
| B-1 and B-2 | Original |
| C-1 thru C-3 | Original |
| C-4 | Blank |
| D-1 | Original |
| D-2 | Blank |
| E-1 | Original |
| E-2 | Blank |
| F-1 | Original |
| F-2 | Blank |
| 1 thru 3 | Original |
| 4 | Blank |

TABLE OF CONTENTS

TABLE OF CONTENTS CONT.

## LIST OF ILLUSTRATIONS



## LIST OF TABLES

# INTRODUCTION

This manual provides descriptive and operational information for the Burroughs Multipoint Communications Service (BMULTI), which enables B 20 and XE 520 systems to communicate with other Burroughs systems using the Burroughs multipoint protocol. The information is presented as follows:

Section 1, Overview
Section 2, Operation
Section 3, Programming Interface
Section 4, Protocol Description

Only the first two sections need to be read by the B 20 or XE 520 operator. Section 3 is required for the programmer implementing a BMULTI application. Section 4 is provided for the convenience of the user.

The appendices provide sample programs and ASCII codes and discuss required hardware. The following technical manuals provide further information:

B 20 Operating Systems Reference Manual
XE 520 System Programmer's Guide

# SECTION 1
# OVERVIEW

## INTRODUCTION

The Burroughs Multipoint Communications Service (BMULTI) allows a B 20 or XE 520 system to communicate with larger Burroughs systems using the Burroughs multipoint protocol. Via BMULTI, a B 20 can be treated, for most purposes, like a standard Burroughs terminal.

In order to use BMULTI, the user must provide an application (such as the B 20 MT983 Emulator) with an interface to BMULTI. A programming interface is available to allow users to develop their own applications, by means of any of the B 20 languages.

## FEATURES

The Burroughs Multipoint Communications Service (BMULTI) includes the following features:

o It supports up to 32 applications, each capable of handling messages of up to 4096 bytes (including protocol control characters).

o It supports synchronous and asynchronous transmission at 110 to 9600 bits per second.

o It supports normal poll, normal select, group poll, fast select, group select, broadcast select, and multipoint contention.

o It includes several options for transmission numbering.

o It allows user configuration of Clear-to-Send, Transmit-to-Receive, and Request-to-Send-Hold delays.

o It can be configured to answer (send EOT) polls to addresses which do not have an application attached.

## SYSTEM REQUIREMENTS

The BMULTI system service requires at least 36 k-bytes of user memory. In addition, the object modules that must be linked with an application require additional memory, which varies with the programmatic interface used. Table 3-5 lists the memory requirements of the different object modules available and the procedures each module serves.

BMULTI will run on an XE 520 cluster processor only if equipped with a memory expansion board.

BMULTI can run on any B 21, B 22, or B 26 workstation (except a B 21-1), and on either the cluster processor or the terminal processor of an XE 520. Hardware requirements are explained in appendix C.

## MAINFRAME NDL IMPACT

BMULTI is inherently slower than a Burroughs terminal, where the processor is completely dedicated to data comm processing. Mainframe NDL timeout values should not be set for a value of less than one second.

The configurable delays offered by BMULTI are upwardly variable because certain processes in the operating system have a higher priority than BMULTI. The actual delay is never less than that with which BMULTI is configured, but it may be more.

# SECTION 2
# INSTALLATION AND OPERATION

## GENERAL

The BMULTI software is available on both 8-inch and 5-1/4 inch diskettes. The 8-inch package is for hard disk installation only on B 22 systems. The 5-1/4 inch package allows both hard disk installation on B 21 and B 25 systems, and supports dual floppy standalone operation. Either package may be used for XE 520 installation. The same procedure is used for installing the package from both sets of diskettes.

## INSTALLATION ON HARD DISK SYSTEMS

Perform the following steps for a successful installation.

1. Login as follows.

```
Command  Path                        press RETURN
Path
  [Volume]                    sys    press RETURN
  [Directory]                 sys    press RETURN
  [Default file prefix]
  [Password]
```

   If your hard disk has a volume password on [d0], type this password into the [Password] field. Press GO.

2. Turn off any cluster workstations.

3. Insert the product distribution diskette, labeled B 20 Poll Select, disk 1, in drive [f0]. (Do NOT press the RESET button.)

4. Install the product as follows:

```
Command  Software Installation   press RETURN
Software Installation
  [Cmd File]
  [Files to]
  [Confirm?]
  [Install file]
```

   Refer to the B 20 System Software Operation Guide to determine whether you should enter any parameters in this form. Press GO.

   It is not recommended that any parameter be entered in the '[Files to]' line of the Software Installation command form.

5. You will be prompted to power down your cluster stations if you have not already done so. Press GO.

6. The message "INSTALLATION OF BMULTI COMPLETE" is displayed when installation is complete. Remove the product distribution diskette and save it as an archive.

7. Turn on your cluster workstations.

# INSTALLATION ON DUAL FLOPPY STANDALONE SYSTEMS

Before the BMULTI release diskette can be used to install the BMULTI service, the following steps must be performed.

1. Boot the system with the system disk in drive [f0]. Replace the system disk with Disk 2 of the Dual Floppy Standalone System Software.

2. Place the BMULTI release diskette in drive [f1].

3. Type the following:

```
Copy
   [f0]<Sys>Exec.run
   [f1]<Sys>Exec.run      Press GO.
```

4. Remove the system disk from drive [f0]. Put the BMULTI release disk in drive [f0] before any BMULTI commands are executed.

# NEW BMULTI COMMANDS

**CONFIGURE BMULTI:** This command has one parameter, [Configuration file], and is used to create and edit BMULTI configuration files. The run file used by this command is "BmFileEdit.run".

**INSTALL BMULTI:** This command has one parameter, [Configuration file], and is used to execute the "BmZip.run" program. This program reads the configuration file (the default is [Sys]<Sys>BmultiConfig.sys), builds a parameter block in long-lived memory, and chains to [Sys]<Sys>Bmulti.run.

**PURGE BMULTI STATION:** This command is used to unlock a locked BMULTI station. No parameters are required to invoke this command. The run file used by this command is "BmPurge.run".

# FILES ON THE DISKETTES

The following files are present on the release diskettes. Both 8-inch and 5-1/4 inch diskettes contain the same files (with two exceptions).

```
[POS5.0-1]<SYS>CrashDump.sys
[POS5.0-1]<SYS>fileHeaders.sys
[POS5.0-1]<SYS>mfd.sys
[POS5.0-1]<SYS>sysImage.sys
[POS5.0-1]<SYS>diagtest.sys      ***
[POS5.0-1]<SYS>bootext.sys       ***
[POS5.0-1]<SYS>Install.sub
[POS5.0-1]<SYS>log.sys
[POS5.0-1]<SYS>sys.cmds          **
[POS5.0-1]<SYS>badBlk.sys
[POS5.0-1]<SYS>FdSys.Version
[POS5.0-1]<SYS>Bmulti.run
[POS5.0-1]<SYS>BmultiConfig.sys
[POS5.0-1]<SYS>BmZip.run
[POS5.0-1]<B20PS5>Bmulti.lib
[POS5.0-1]<B20PS5>BmPurge.run
[POS5.0-1]<B20PS5>BmFileEdit.run
```

**Present only on 5 1/4 inch diskettes.
***Present only on 8 inch diskettes.

BMULTI requires approximately 36 k-bytes of memory.

# CONFIGURING THE BMULTI SERVICE

To configure the communications service, type "Configure Bmulti" in the command field of the command form (refer to the B 20 Systems Executive Reference Manual). The form illustrated below then appears.

```
Configure Bmulti
   [Configuration file] _____________________________
```

The default is [Sys]<Sys>BmultiConfig.sys.

When executed, the screen is cleared and the following form is displayed (assuming that the program was executed with the default parameter):

```
BMULTI CONFIGURATION FILE EDITOR 5.0

Currently Open File:  [Sys]<Sys>BmultiConfig.sys
```

```
 Open  | Save  |Discard        |     |     |          |Modify |Modify
Config|Config|Config          |     |     |          |Virtual|Bmulti
 File  | File  | File          |     |     |          | Addrs |Params
```

This is the basic entry state. To modify the BMULTI parameters which affect every station, press f10. To examine or modify the list of virtual addresses, press f9. Press f2 in order to close and save the currently open configuration file. Press f1 to open a new one. Press f3 in order to close and not save the currently open configuration file.

If f1 is pressed, the following is displayed:

```
    Config File name _____________________________
```

Enter the name of the configuration file you wish to edit, such as [sys]<sys>BmultiConfig.sys, and press GO. If the file is already present, the utility will display "Opening ..". If it is not already present, the utility will display "Opening .. a NEW file ..." and do so. In either case, the utility will display "Done" when it is finished and has returned to the basic entry state.

If f2 is pressed, the utility will display "Saving ..". When the utility has closed the file, it will display "Done" and return to the basic entry state.

If f3 is pressed, the utility will display "Discarding ..". When the utility has closed the file, it will display "Done" and return to the basic entry state.

If f9 is pressed, the prompts on the line of softkeys changes to the following:

```
Add |Delete| List          |     |     |       |       | Exit
```

Pressing f10 returns the utility to the basic entry state. Pressing f3 causes the utility to list all currently configured virtual addresses. Pressing f1 causes the utility to prompt for a virtual address to be added to the list; pressing f2 causes the utility to prompt for an address to be deleted from the list.

If f10 is pressed, the utility displays "Modify Bmulti Options" in reverse video and then prompts, in order, for the ten BMULTI parameters as follows:

```
Group Poll Address      : _
Group Select Character  : _
Sync or Async?          : _
Channel        [A/D]    : _
Baud Rate [max 9600]    : ____
Xmno option    [0..5]   : _
CTS delay      [0..255] : ___
Xmt-Rcv delay  [0..255] : ___
RTS Hold       [0..255] : ___
Downstream Station?     : _
```

After each parameter is entered, press RETURN to go on to the next. The utility displays "OK" if each value is acceptable. When all ten are entered, the program returns to the basic entry state.

The BMULTI parameters are discussed in more detail below:

Group Poll Address
: is the group poll address to be used by BMULTI for all of its stations. A cluster system may have only one group poll address. This must be any two ASCII characters between 020h and 07Fh. If group poll is not used, any address not polled may be used.

Group Select Character
: is the character which is recognized as the group select character to be used by BMULTI for all of its stations. A cluster system may have only one group select character. This must be any ASCII character between 020h and 07Fh. If group select is not used, any character not already in use as a poll or select character may be used.

Sync or Async?
: If this is answered sync, clocking is to be provided by the modem. If answered async, the B 20 supplies its own clocking.

Channel [A..D]
: On a B 20 workstation, this is either A or B. On an XE 520 Cluster Processor, this is either A or B. On an XE 520 Terminal Processor, this is A, B, C, or D.

Baud Rate [max 9600]
: must be one of the following: 110, 150, 300, 600, 1000, 1200, 1800, 2000, 2400, 4800, or 9600, except on an XE 520, where 110 and 150 are not permissable entries. These are the permissable transmission speeds in bits per second.

Xmno option [0..5]
: is 0, 1, 2, 3, 4, or 5. Each of these numbers stands for a particular transmission numbering scheme. 0 is for no transmission numbers. 1 indicates an alternating zero and one scheme while 2 means an alternating @ and A (TD830 compatibility). Scheme 3

is a modulus 10 (0 through 9, wrapping around), 4 is modulus 100, and 5 is modulus 1000. (Refer to section 4 of the BMULTI Reference Manual for more information.)

CTS delay [0..255]
is the Clear-to-send delay in milliseconds. BMULTI waits this period of time after turning on Request-to-send before looking for Clear-to-send from the modem. If Clear-to-send is not on when the timer expires, BMULTI waits until it does go on before transmission.

Xmt-Rcv delay [0..255]
is the Receive delay in milliseconds. When BMULTI turns off Request-to-send after a transmission, it waits this period of time before examining incoming data. This delay is normally non-zero when using Burroughs Two-wire Direct Interface (TDI).

RTS Hold [0..255]
is the period of time, in milliseconds, which BMULTI keeps Request-to-send on after the end of a transmission. This delay is used with some older modems or to cause the host system modem to keep Data Carrier Detect on long enough to ensure that the host receives the transmission.

Downstream Station?
is either "y" or "n". If "y" (yes), BMULTI does not reply to a group poll when Secondary Receive Data is on. If "n" (no), BMULTI ignores Secondary Receive Data when determining its response to a group poll. The response should be YES only when the B 20 running BMULTI is in the midst of a concatenated string of terminals communicating through a single modem. If the B 20 is either the only terminal connected to its modem, or if it is the last terminal on a concatenation string, the response should be NO. It should also be NO when using TDI. This parameter is not used by the XE 520.

# INSTALLING THE BMULTI SYSTEM SERVICE

BMULTI is a system service which may be installed on B 20 standalone systems, the master workstation of B 20 cluster systems, or a cluster processor or terminal processor of an XE 520 system.

To invoke the communications service from the Executive, type "Install Bmulti" in the command field of the command form. The form illustrated below then appears.

Install Bmulti
[Configuration file] _____

To invoke the communications service during system initialization, the system administrator should add the following command line to the appropriate workstation's SysInit.JCL file, cluster processor's InitCpnn.JCL file, or terminal processor's InitTpnn.JCL file:

```
$RUN [Sys]<Sys>BmZip.run(, <configuration file>)
```

where the parentheses enclose optional text.

**NOTE**

In order for BMULTI to take control of a specified channel, that channel must not be already under the control of another program. On an XE 520, no "ASYNC <channel number>" statement may reference that channel in the appropriate configuration file (Cpnn.cnf for the cluster processor and Tpnn.cnf for the terminal processor). On any system, the spooler may not be configured to use the same channel simultaneously.

# PURGING A BMULTI STATION

**NOTE**

Disable the ACTION-FINISH command if you are running on a master or standalone system. Using ACTION-FINISH to terminate your application may cause a system crash.

Occasionally it happens that a BMULTI application is terminated with ACTION-FINISH, or a cluster station running a BMULTI application is powered off accidentally. When this happens, most of the time BMULTI automatically realizes that one of its stations is no longer active, and make that station address available for other application (or for the same one if re-executed).

Under some circumstances, however,BMULTI fails to make such an address available again. If this happens, use the "Purge Bmulti Station" command to force a BMULTI station address to be available. To execute this utility, type "Purge BMULTI Station" in the command field of the command field form. There are no parameters to this command; just press GO.

When executed, the screen is cleared and the following is displayed:

```
BMULTI STATION PURGER  5.0

Enter Station Address  __
```

Enter a two-character station address. If the station is successfully purged, the utility displays "*** Station successfully cleared". If unsuccessful, the station displays "*** Clear unsuccessful".

The utility then displays:

"Hit <FINISH> to exit or any key to continue"

Pressing FINISH causes the program to exit to the Executive. Pressing any other key returns the program to the initial prompt above.

# SECTION 3
# PROGRAMMING INTERFACE

## GENERAL

The user requires application software in order to make use of BMULTI. This program must be written using one of the procedural interfaces detailed below. There are four procedural interfaces available to the BMULTI user: a single-task interface, a multiple-task interface, an enhanced low-level interface and a high-level interface suitable for use by application programmers.

**NOTE**

It is highly recommended that any software development in either COBOL or interpreted BASIC make use of the high-level interface. Details of this interface can be found starting on page 3-41.

The user program, after being written and compiled, must be linked with Bmulti.lib. (Basic Interpreter or Cobol applications require that the interpreter have been linked with Bmulti.lib.)

## COORDINATION

Because the operation of an application system using these facilities is under control of the host computer, there must always be a high degree of coordination between the B 20 application system and the application system on the host computer. There must be agreement on order of procedures, such as which system sends first, and so on. This involves coordination between the application programmers of each system.

# COMMANDS AND REPORTS

A BMULTI application informs BMULTI what it desires to do by means of Commands. BMULTI informs the application of events on the line by means of Reports. Commands from an application may be accepted or denied by BMULTI, depending on the state in which the application's address is. An address may move from one state to another in response to a command or an event on the line. Figure 3-1 is a diagram of BMULTI states.

The application issues commands and obtains reports by issuing an OS primitive called a Request (refer to the BTOS Reference Manual). The application may either suspends itself until a reply is available (Wait) or continues to process, checking periodically to see whether a reply is available (Check).

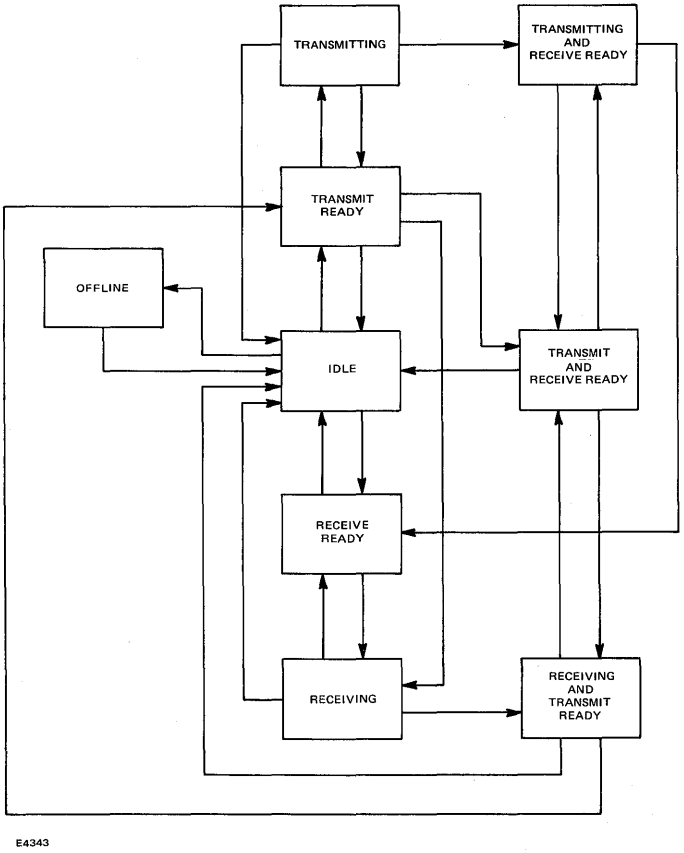



Figure 3-1. BMULTI States

# OVERVIEW OF PROCEDURAL INTERFACE

## Installing BMULTI

The communications service must first be installed, as described in section 2.

## Reserving a Station Address

The DcConfig call (or a Configure command) must be made first to reserve a station address. Until the application system issues a DcConfig call, any attempts by the host computer to select or poll the station are ignored by BMULTI. BMULTI continues to ignore any such attempts until an Online command is issued. When a DcConfig call is accepted by BMULTI, the application system is in the OFFLINE state.

## Online, Offline and Idle

An Online command, when accepted, moves the application from the OFFLINE state to the IDLE state. In this state BMULTI replies to selects and polls addressed to the application station. BMULTI NAKs selects and sends EOT in reply to polls. An Offline command may be used to cause BMULTI to return to the OFFLINE state.

## Transmitting

The command Transmit Ready, when accepted, moves the application to the TRANSMIT READY state. In this state BMULTI looks for the next poll to the applications address. Upon seeing such a poll BMULTI issues a Ready for Transmit Buffer report (4). When returned, the application immediately issues a Transfer Transmit Buffer command to instruct BMULTI to obtain and transmit the buffer. When BMULTI sees an ACK from the host, it returns a Transmit Done report (5). If the application sees a Transmit Error report, it looks for another Ready for Transmit Buffer report and is prepared to issue another Transfer Transmit Buffer command.

## Receiving

The command Receive, when accepted, moves the application to the RECEIVE READY state. In this state BMULTI ACKs any subsequent selects and receives the transmitted message. If BMULTI does not detect an error in the message, it will issue a Receive Done report (3). The reports Receive Error, Duplicate Sequence Number, and Sequence Number Error are issued by BMULTI, instead of Receive Done, when it has received a message in which it has detected an error. The application program is then allowed a reasonable amount of time to issue a Transfer Receive Buffer command to retrieve the buffer. If a time-out occurs before the command is issued, the command is denied. If the command is accepted, an ETX occurs after the last text character in the buffer.

## Idle and Abort

After a Receive or Transmit command has been issued, the Idle command can be used to return the application to the Idle state. However, if a select has already been ACKed or a transmission in response to a poll begun, the Abort command must be used to cause BMULTI to abandon an attempt to receive a message or to transmit a message. (For instance, if a message sent to the B 20 contains imbedded ETXs, BMULTI does not ACK it. DcReport returns a value of 7 several times, indicating a Receive Error. After a number of these in succession, the application issues an Abort command and either warns the operator or ends the session.)

## Receiving Fast, Group and Broadcast Selects

If the Set Fast Ready command is not issued, BMULTI NAKs any fast, group, or broadcast selects addressed to the application's station except when Receive Ready. If the Set Fast Ready command is issued, BMULTI ACKs and receives any such selects, even if not Receive Ready. (During the reception of a group or broadcast select, DcReport returns a value of 2 in order to distinguish between messages designated for that station in particular and messages designated for many stations.) After a Set Fast Select command has been issued, a Reset Fast Select command may be issued to prevent reception of fast, group, or broadcast selects.

## Terminating

**NOTE**

Disable the ACTION-FINISH command if you are running on a master or standalone system. Using ACTION-FINISH to terminate your application may cause a system crash.

The application terminates a communication session by issuing an End Session command. This command is accepted only from the IDLE and OFFLINE states.

## Command Error Return Codes

The Error Return Code should be checked following each DcConfig and DcCommand call. Unpredictable results may occur if the program assumes that a command was accepted when it was in fact rejected.

## Report Queue

BMULTI maintains a 10-deep queue of reports for each active address. The report codes Receiving, Receiving Group or Broadcast Select, Select Denied, and Receive Error are added to the Report queue only if each is not already present in the queue. It is good programming practice to keep the Report Queue as shallow as possible by reading it frequently; if the Report Queue is too deep, reports returned by BMULTI may be obsolete.

Sample programs using the BMULTI system service procedures are listed in Appendix A. The programs presented are in FORTRAN, COBOL, Pascal, and BASIC.

# PROCEDURES FOR SINGLE-TASK INTERFACE

## DcConfig

### Description

The DcConfig procedure passes a station address from the application to BMULTI. If the address has not been assigned by another application on the system, and the maximum number of applications will not be exceeded, BMULTI acknowledges the assignment; otherwise, it indicates an error. Table 3-1 contains the values which the error return code can take.

### Procedural Interface

```
DcConfig (devAddr): ErcType
```

where

devAddr is a word containing two ASCII characters. This address must not duplicate that of any other stations on the same line.

### Request Block

| Offset | Field | Size (bytes) | Contents |
|---|---|---|---|
| 0 | sCntInfo | 2 | 3 |
| 2 | nReqPbCb | 1 | 1 |
| 3 | nRespPbCb | 1 | 1 |
| 4 | userNum | 2 | |
| 6 | exchResp | 2 | |
| 8 | ercRet | 2 | |
| 10 | rqCode | 2 | FFFF |
| 12 | subrqcode | 1 | 0 |
| 13 | devadr | 2 | |
| 15 | pCpBuffx | 4 | |
| 19 | sCpBuffx | 2 | |
| 21 | pCpBuffr | 4 | |
| 25 | sCpBuffr | 2 | |

## DcCommand

### Description

The DcCommand procedure is used to pass commands from the application to BMULTI. Table 3-1 contains the values which the returned integer can take.

### Procedural Interface

```
DcCommand (command, pBuffer, sBuffer): ErcType
```

where

command — is a word containing a value from table 3-2.

pBuffer
sBuffer — describe the application's message buffer. While transferring a receive buffer, sBuffer must be at least one byte larger than the largest message to be received from the host system (the extra byte is used to store ETX). When transferring a transmit buffer, sBuffer must be the exact size of the message to be transmitted. These two parameters must be specified for every command even though only two of the commands use them.

### Request Block

DcCommand uses the same request block as DcConfig. In both cases the Request call is used to pass the Request block and the Wait call is used to receive the reply. The distinction between the calls lies in the subrequest code. The DcConfig call uses the subrequest code of zero. The DcCommand call uses the command code (see table 3-2) as the subrequest code.

**Table 3-1: Error Return Codes : Single-Task Interface**

System error = 0

Returned by the Transfer Receive Buffer command when the buffer is larger than the size passed by the application. Also returned by all commands when an unforeseen BTOS error has occurred.

Command accepted = 1

Command denied = 2

Returned when an invalid state transition is requested. Also returned by Transfer Receive Buffer when a time-out has occurred and the buffer is no longer available.

**Table 3-2: Command Codes**

| | |
|---|---|
| Transfer Receive Buffer | = 1 |
| Transfer Transmit Buffer | = 2 |
| Offline | = 3 |
| Online | = 4 |
| Idle | = 5 |
| Set Fast Ready | = 6 |
| Receive | = 7 |
| Transmit Ready | = 8 |
| End Session | = 9 |
| Abort | = 10 (or 0Ah) |
| Reset Fast Ready | = 11 (or 0Bh) |

## DcReport

### Description

The DcReport procedure returns the status of the datacomm subsystem to the application. This procedure is the means by which BMULTI informs the application of events on the line.

### Procedural Interface

```
DcReport: Integer
```

### Request Block

| Offset | Field | Size (bytes) | Contents |
|---|---|---|---|
| 0 | sCntInfo | 2 | 2 |
| 2 | nReqPbCb | 1 | 0 |
| 3 | nRespPbCb | 1 | 1 |
| 4 | userNum | 2 | |
| 6 | exchResp | 2 | |
| 8 | ercRet | 2 | |
| 10 | rqCode | 2 | FFFE |
| 12 | devadr | 2 | |
| 14 | pReport | 4 | |
| 18 | sReport | 2 | 1 |

The DcReport call performs a Request primitive, passing this request block, then performs a Check primitive to receive the reply. If no report is forthcoming, DcReport returns a zero. Care must be taken when programming at the Request/Check level; reports may be queued, and the first Report available at any one time may not reflect the true state of the line.

The reports returned are as follows:

Table 3-3. Report Return Codes

No Report = 0

Returned by DcReport if no report is available.

Receiving = 1

Returned if SOH detected for Receive-Ready address.

Receiving Group or Broadcast select = 2

Returned if a group or broadcast select has been addressed to a fast-select-ready application.

Receive Done = 3

Returned if an error-free message has been received.

Ready for Transmit Buffer = 4

Returned when a poll has been received and the application was transmit-ready.

Transmit Done = 5

Returned when an ACK has been received after a transmission to the host.

Select Denied = 6

Returned if a Fast, Group, Broadcast, or regular select has been NAKed.

Receive Error = 7

Returned if a block check or parity error occurs in a message addressed to the application.

Duplicate Sequence Number = 8

Returned if a duplicate transmission number is detected (if transmission numbers are enabled), in place of Receive Done.

Sequence Number Error = 9

Returned if a transmission number out of sequence is detected (if non-alternating transmission numbers are enabled), in place of Receive Done.

Transmit Error = 10 (0Ah)

Returned if an EOT is received in response to a transmission or if an Abort command is issued.

## DcReportWait

### Description

The DcReportWait procedure is identical to the DcReport procedure except that a report of zero is never returned. The procedure waits until a non-zero report is available before returning. The reports are the same as for DcReport.

### Procedural Interface

```
DcReportWait: Integer
```

### Request Block

DcReportWait uses the same request block that DcReport does, but it uses a Wait call to receive the reply instead of a Check. Care must be taken when programming at the Request/Wait level; reports may be queued, and the first report available at any one time may not reflect the true state of the line.

# PROCEDURES FOR MULTI-TASK INTERFACE

The Multiple-Task Interface is similar to the programming interface presented above. However, it allows up to three different addresses to be used per application. In addition, DcConfig has been replaced by the MpCommand Configure, and the error codes returned by MpCommand, MpReport, and MpReportWait are different. The report codes and command codes are the same.

**NOTE**

The two procedural interfaces may not be mixed; for example, an application using DcCommand must use DcConfig and DcReport (or DcReportWait): it may not use any of the multiple-task procedures.

## MpCommand

### Description

The MpCommand procedure is used to pass commands from the application to BMULTI. Table 3-3 contains the values which the returned integer can take.

### Procedural Interface

```
MpCommand (command, devAdr, pBuffer,
    sBuffer): Erctype
```

where

command — is a word containing a value from table 3-2, or zero for the Configure command.

devAdr — is a word containing the BMULTI address.

pBuffer
sBuffer — describe the application's message buffer. While transferring a receive buffer, sBuffer must be at least one byte larger than the largest message to be received from the host system (the extra byte is used to store ETX). When transferring a transmit buffer, sBuffer must be the exact size of the message to be transmitted. These two parameters must be specified for every command even though only the two transfer commands use them.

**Request Block**

MpCommand uses the same request block that DcConfig and DcCommand do.

**Table 3-4: Error Return Codes for Multiple-Task Interface**

| ERROR CODE (HEX) | ERROR CODE (DECIMAL) | EXPLANATION |
|---|---|---|
| 0000 | 0 | Command or Report accepted. |
| 8000 | 32768 | Invalid Command. Returned by MpCommand when a command code is greater than 0Bh. |
| 8001 | 32769 | Task Overflow. The maximum number of addresses has been reached. Returned by the Configure command. |
| 8002 | 32770 | Command Pending. There is already a command pending for this address. Returned by MpCommand. |
| 8003 | 32771 | Report Pending. Returned by MpReportWait when a report request is already pending for this address. |
| 8004 | 32772 | Invalid Address. Returned by MpCommand and MpReport when the device address has not already been configured. |
| 8005 | 32773 | Command Denied. Returned by MpCommand when an invalid state transition is requested. |
| 8006 | 32774 | Buffer Overflow. Returned by MpCommand when a Transfer Receive Buffer command passes an sBufferMax value smaller than the length of the data received from the host. |
| 8007 | 32775 | Report Error. Returned by MpReport and MpReportWait when BMULTI returns a non-zero value in the ercRet field of the Report request block. |

## MpReport

### Description

The MpReport procedure returns the status of the datacomm subsystem to the application. This procedure is the means by which BMULTI informs the application of events on the line.

### Procedural Interface

```
MpReport (devAdr, pReportRet): ErcType
```

where

devAdr — is a word containing the BMULTI address.

pReportRet — is a pointer to a word to which the Report is to be returned by BMULTI.

### Request Block

MpReport uses the same request block that DcReport and DcReportWait do. After issuing the request, MpReport performs a Check primitive to receive the reply. If no report is forthcoming, MpReport returns a zero.

## MpReportWait

### Description

The MpReportWait procedure is identical to the MpReport procedure except that a report of zero is never returned. The procedure waits until a non-zero report is available before returning. The reports are the same as for DcReport, DcReportWait, and MpReport.

### Procedural Interface

```
MpReportWait (devAdr, pReportRet) : ErcType
```

where

| | |
|---|---|
| devAdr | is a word containing the BMULTI address. |
| pReportRet | is a pointer to a word into which the Report is to be returned by BMULTI. |

### Request Block

MpReportWait uses the same request block that DcReport, DcReportWait, and MpReport do, but it waits to receive the reply instead of timeing out if a report is not available. Care must be taken when programming at the Request/Wait level; reports may be queued, and the first report available at any one time may not reflect the true state of the line.

# BMULTI STATE MACHINE

BMULTI can be thought of as a state machine: that is, the action that BMULTI takes in response to a command from an application varies depending upon the current state of BMULTI, which depends in turn upon previous commands and events on the data communications line (which are passed to the application as reports). BMULTI may be in any one of nine states (see figure 3-1 for details). BMULTI runs a parallel state machine for each address. The section below describes each of the nine states in detail.

## Command Accepted and Command Denied

In the following discussion, the term "accepted" is used to indicate that BMULTI returns a Command Accepted code and performs the requested action. Otherwise, BMULTI "denies"; which means that it returns a Command Denied code and does not perform the requested action.

## Fast Ready Flag

The Set Fast Ready and Reset Fast Ready commands are accepted in all states except the Offline state. These two commands alter the setting of internal flag which BMULTI uses to determine the appropriate response to Fast, Group, and Broadcast selects.

## End Session Command

The End Session command is accepted from the Offline state and the Idle state. It is denied from all other states. The End Session command causes BMULTI to remove the associated address from the table of currently active addresses.

## Offline State

This is the initial state of any address when BMULTI accepts a Configure command. In this state, BMULTI ignores all control sequences for the assigned address.

The following command is accepted. All other commands, except End Session, are denied.

Online:
    State change: Idle

## Idle State

In this state BMULTI is not ready to receive any data, but is responsible for responding for the address. However, Fast, Group, and Broadcast selects are accepted if the Fast Ready flag is on.

In this state, the following commands are accepted. The others are denied.

Offline:

State change: Offline

Idle:

State change: none

Receive:

State change: Receive Ready

Transmit:

State change: Transmit Ready

Abort:

State change: none

Inputs from the communications channel:

Poll of configured address:

| | |
|---|---|
| State change: | none |
| Report: | none |
| Action: | Transmit an EOT |

Group Poll of installed group poll address:

| | |
|---|---|
| State change: | none |
| Report: | none |
| Action: | If downstream RTS is false and none of the other addresses assigned to this cluster are in transmit ready, then transmit an EOT; otherwise no response. |

Select of configured address:

| | |
|---|---|
| State change: | none |
| Report: | Select Denied |
| Action: | Transmit a NAK |

Fast Select of configured address with Fast Ready flag set:

| | |
|---|---|
| State change: | Receiving |
| Report: | Receiving |
| Action: | Wait for SOH |

Fast Select of configured address with Fast Ready flag reset:

| | |
|---|---|
| State change: | None |
| Report: | Select Denied |
| Action: | Wait for the ETX and transmit a NAK. |

Broadcast Select of configured address or Group Select of configured address with installed group select character, with Fast Ready flag set:

State change: Receiving

Report: Receiving Group or Broadcast Select

Action: Wait for SOH

Broadcast Select of configured address or Group Select of configured address with installed group select character, Fast Ready flag reset:

State change: None

Report: Select Denied

Action: Wait for the ETX and transmit a NAK.

Broadcast Select of non-configured address or Group Select of non-configured address (any group select character), with Fast Ready flag reset:

State change: None

Report: None

Broadcast Select of non-configured address or Group Select of non-configured address (any group select character), with Fast Ready flag set:

State change: Receiving

Report: Receiving Group or Broadcast Select

### Transmit Ready State

In this state BMULTI is ready to transmit and is not ready to receive any data. The following commands are accepted. The others are denied.

Idle:

State change: Idle

Receive:

State change: Transmit and Receive Ready

Transmit:

State change: none

Abort:

State change: Idle

Inputs from the Communications channel:

Poll of configured address:

State change: Transmitting

Report: Ready for Transmit Buffer

Action: Transmit message

Group Poll of installed group poll address:

State change: Transmitting

Report: Ready for Transmit Buffer

Action: Block downstream RTS and CTS; transmit message for each online application which is transmit ready, one by one.

Select of configured address:

State change: None

Report: Select Denied

Action: Transmit a NAK

Fast Select of configured address with Fast Ready flag set:

State change: Receiving and Transmit Ready

Report: Receiving

Action: Wait for SOH

Fast Select of configured address with Fast Ready flag reset:

State change: None

Report: Select Denied

Action: Wait for ETX and Transmit a NAK.

Broadcast Select of configured address or Group Select of configured address with installed group select character, with Fast Ready set:

State change: Receiving and Transmit Ready

Report: Receiving Group or Broadcast Select

Action: Wait for SOH

Broadcast Select of configured address or Group Select of configured address with installed group select character, with Fast Ready reset:

State change: None

Report: None

Action: Wait for ETX and transmit a NAK

Broadcast Select of unconfigured address or Group Select of unconfigured address with installed group select character, with Fast Ready set:

State change: Receiving

Report: Receiving Group or Broadcast Select

## Transmitting State

In this state BMULTI has recognized a poll or group poll and is ready to transmit data on the communications channel.

The following commands are accepted; the others are denied.

Idle (if CTS is not ON):

State change: Idle

Action: Turn off RTS

Receive:

State change: Transmitting and Receive Ready

Abort:

State change: None

Action: Set Fast Ready to False
Turn off RTS

Inputs from the Communications channel:

EOT:

State change: Transmit Ready

Report: None

Action: Unblock downstream RTS and CTS

ACK:

State change: Idle

Report: Transmit Done

Action: If no more applications are Transmit Ready (for a group poll) then unblock downstream RTS and CTS. If downstream RTS is false then transmit EOT. For specific poll, transmit EOT.

NAK:

State change: None

Report: None

Action: Retransmit the data according to the protocol.

RVI:

If specific poll:

State change: Idle

Action: Transmit an EOT.

If station transmitted last in reply to a group poll:

State change: Idle

Action: Unblock downstream RTS and CTS. If downstream RTS is false transmit an EOT, otherwise no response.

If station is waiting to be unblocked and has not had an opportunity to reply to the group poll:

State change: Transmit Ready

Report: Transmit Error

Action: Unblock downstream RTS and CTS. If downstream RTS is false transmit EOT otherwise no response.

## Transmitting and Receive Ready State

In this state BMULTI has recognized a poll or group poll and is ready to transmit data on the communication channel. The addressed workstation is also ready to accept data and when the transmission is complete, the workstation will be in the Receive Ready state.

The following commands are accepted. The others are denied.

Idle (if CTS is not ON):

State change: Idle

Action: Turn off RTS

Receive:

State change: None

Abort:

State change: Idle

Action: Set Fast Ready to False
Turn off RTS

Inputs from the communications channel:

EOT:

State change: Transmit and Receive Ready

Report: None

Action: Unblock downstream RTS and CTS.

ACK: (for cluster stations individually in case of group poll)

State change: Receive Ready

Report: Transmit Done

NAK:

State change: None

Report: None

Action: Retransmit the data.

RVI:

If specific poll:

State change: Idle

Action: Transmit an EOT.

If station transmitted last in reply to a group poll:

State change: Receive Ready

Report: Transmit Done

Action: Unblock downstream RTS and CTS. If downstream RTS is false transmit an EOT, otherwise no response.

If station is waiting to be unblocked and has not had an opportunity to reply to the group poll:

State change: Transmit Ready and Receive Ready

Report: Transmit Error

Action: Unblock downstream RTS and CTS. If downstream RTS is false transmit an EOT otherwise no response.

## Transmit and Receive Ready State

In this state the protocol handler is ready to transmit and to receive data. The following commands are accepted.

Idle:

State change: Idle

Receive:

State change: None

Transmit:

State change: None

Abort:

State change: Idle

Inputs from the communications channel:

Poll of configured address:

State change: Transmitting and Receive Ready

Report: Ready for Transmit Buffer

Action: Block downstream RTS and transmit data

Group Poll of installed group poll address:

State change: Transmitting and Receive Ready

Report: Ready for Transmit Buffer

Action: Block downstream RTS and CTS

Select of configured address:

State change: Receiving and Transmit Ready

Report: Receiving

Fast Select of configured address:

State change: Receiving and Transmit Ready

Report: Receiving

Action: Wait for SOH

Broadcast Select:

State change: Receiving and Transmit Ready

Report: Receiving

Action: Wait for SOH

## Receiving State

In this state the protocol handler is receiving a block of data. The command which is accepted is:

Abort:

State change: Idle

Action: Set Fast Ready to False
Turn off RTS

Inputs from the communications channel:

EOT:

State change: Receive Ready

Report: None

ETX (and Block Check Character)
(no Parity or BCC error):

State change: Idle

Report: Receive Done

Action: Transmit and ACK if select was on my address

ETX (and Block Check Character)
(Parity or BCC error):

State change: None

Report: None

Action: Send NAK if select was on my address and wait for SOH or EOT.

### Receive Ready State

In this state BMULTI is ready to receive data. Commands accepted are:

Idle:

State change: Idle

Receive:

State change: None

Abort:

State change: Idle

Inputs from the communications channel:

Poll of configured address:

State change: None

Report: None

Action: If downstream RTS is False then transmit an EOT

Group Poll of configured group poll address:

State change: None

Report: None

Action: If downstream RTS is False then transmit an EOT.

Select of configured address:

State Change: Receiving

Report: Receiving

Fast Select of configured address:

State change: Receiving

Report: Receiving

Action: Wait for SOH

Broadcast Select:

State change: Receiving

Report: Receiving

Action: Wait for SOH

Group Select of installed group select character:

State change: Receiving

Report: Receiving

Action: Wait for SOH

## Receiving and Transmit Ready State

In this state BMULTI is receiving a block of data and is also ready to transmit on the next poll with this workstation's address. Commands accepted are:

Idle:

State change: Idle

Abort:

State change: Idle

Action: Set Fast Ready to False

Inputs from the communications channel:

EOT:

State change: Transmit and Receive Ready

Report: None

ETX (and Block Check Character)
(No parity or BCC errors):

State change: Transmit Ready

Report: Receive Done

Action: Transmit an ACK if select was on my address

ETX (and Block Check Character)
(Parity or BCC errors):

State change: None

Report: None

Action: Send NAK if select was on my address and wait for SOH or EOT.

# LOW-LEVEL INTERFACE PROCEDURES

## BmOpen

### Description

The BmOpen procedure is the first called by the application. It passes a station address to Bmulti and returns a Task Handle to be used by the application when calling other procedures.

### Procedural Interface

```
BmOpen (devadr, pTskH, fSys) : Erctype
```

where

devad is a word containing two ASCII characters. This address must not duplicate that of any other station on the same line.

pTskH is a pointer to a byte.

fSys is a byte or Boolean. It should be set to TRUE if the application making the call is to be a system service.

### Request Block

| Offset | Field | Size (bytes) | Contents |
|---|---|---|---|
| 0 | sCntInfo | 2 | 4 |
| 2 | nReqPbCb | 1 | 1 |
| 3 | nRespPbCb | 1 | 1 |
| 4 | userNum | 2 | |
| 6 | exchRet | 2 | |
| 8 | ercRet | 2 | |
| 10 | rqCode | 2 | FFFD |
| 12 | subrqCode | 1 | 0 |
| 13 | auxInfo | 1 | |
| 14 | DevHandle | 2 | |
| 16 | pCpBuffx | 4 | |
| 20 | sCpBuffx | 2 | |
| 22 | pCpBuffr | 4 | |
| 26 | sCpBuffr | 2 | |

## BmCommand

**Description**

The BmCommand procedure is used to pass commands from the application to Bmulti.

**Procedural Interface**

```
BmCommand (TskH, Command, pBuff, sBuff) : Erctype
```

where

TskH is a byte (this value is returned by BmOpen).

Command is a word containing a value from the table below.

pBuff
sBuff describe the application's message buffer. These are normally dummy values except for the two buffer transfer commands (1 and 2). For Get Received Buffer, sBuff is set to the maximum number of bytes which the application can accept. After BmCommand returns, the first two bytes starting at pBuff will contain the number of bytes received. For Send Transmit Buffer, sBuff is set to the actual number of bytes the application wishes to transmit.

Possible values of Command are :

```
 1   Get Received Buffer*
 2   Send Tranmsmit Buffer*
 3   Offline
 4   Online
 5   Idle
 6   Set Fast Ready
 7   Set Receive Ready
 8   Set Transmit Ready
 9   End session
10   Abort
11   Reset Fast Ready
12   Transmit Extended Message* (Messages > 2048)
```

* These commands require valid pBuff and sBuff.

**Request Block**

BmCommand uses the same request block as BmOpen.

## BmReport

**Description**

The BmReport procedure returns the status of the datacomm subsystem to the application. This procedure is the means by which BMULTI informs the application of events on the line. This procedure returns a report of zero if no report is available. The report is valid only if the procedure returns zero.

**Procedural Interface**

```
BmReport (TskH, pReport): Erctype
```

where

TskH is a byte (this value is returned by BmOpen).

pReport is a pointer to a word where the procedure is to return the report.

**Request Block**

| Offset | Field | Size (bytes) | Contents |
|---|---|---|---|
| 0 | sCntInfo | 2 | 2 |
| 2 | nReqPbCb | 1 | 0 |
| 3 | nRespPbCb | 1 | 1 |
| 4 | userNum | 2 | |
| 6 | exchRet | 2 | |
| 8 | ercRet | 2 | |
| 10 | rqCode | 2 | FFFC |
| 12 | DevHandle | 2 | |
| 14 | pReportNo | 4 | |
| 18 | sReportNo | 2 | 1 |

## BmReportWait

**Description**

This procedure waits for a report. The report is valid only if the procedure returns zero.

**Procedural Interface**

```
BmReportWait (TskH, pReport) : Erctype
```

where

TskH is a byte (this value is returnd by BmOpen).

pReport is a pointer to a word where the procedure is to return the report.

The reports that can be returned by BmReport and BmReportWait are:

0 No report
1 Receiving
2 Receiving Group Or Broadcast Select
3 Receive Done
4 Ready for Transmit buffer
5 Transmit Done
6 Select Denied
7 Receive Error
8 Duplicate Transmission Number
9 Transmission Number Error
10 Transmit Error

**Request Block**

BmReportWait uses the same request block as BmReport.

## BmReportTimeout

**Description**

This procedure waits for a report for a specified period of time. The report is valid only if the procedure returns zero.

**Procedural Interface**

```
BmReportTimeout (TskH, pReport, timeout): Erctype
```

where

TskH is a byte (this value is returnd by BmOpen).

pReport is a pointer to a word where the procedure is to return the report.

timeout is a word, giving an interval (in tenths of a second during which the procedure will wait for a report.

The reports that can be returned by BmReportTimeout are the same as those that can be returned by BmReport and BmReportWait.

**Request Block**

BmReportTimeout uses the same request block as BmReport and BmReportWait.

## BmIdentify

### Description

This procedure provides some information about the version of Bmulti currently running. The information is returned into a user-provided buffer which should be 14 bytes long. The first two bytes (a word) will contain the workstation number. The third byte will contain the channel in ASCII ('A' or 'B'). The last 11 bytes will contain the version number of Bmulti in the form of a Pascal lstring (out of these 11 bytes, the first byte will contain the actual number of characters in the version number).

### Procedural Interface

```
BmIdentify (pIdBlk, sIdBlk): ErcType
```

where

pIdBlk
sIdBlk describe the Status Block

### Request Block

| Offset | Field | Size (bytes) | Contents |
|---|---|---|---|
| 0 | sCntInfo | 2 | 0 |
| 2 | nReqPbCb | 1 | 0 |
| 3 | nRespPbCb | 1 | 1 |
| 4 | userNum | 2 | |
| 6 | exchRet | 2 | |
| 8 | ercRet | 2 | |
| 10 | rqCode | 2 | FFFB |
| 12 | pCpBuffr | 4 | |
| 16 | sCpBuffr | 2 | 14 |

## BmQuery

**Description**

This procedure can be used by an application to obtain information about BMULTI's internal variable status. The application supplies a buffer into which BMULTI returns the status. This buffer size should be at least 81 bytes. The maximum size of the status buffer should be passed in a variable whose address is supplied as a parameter to this procedure. After the procedure returns, this variable contains the actual number of bytes returned.

**Procedural Interface**

BmQuery (pStat, psStat): Erctype

where

pStat is a pointer to the buffer where status information is to be returned.

psSta is a pointer to the word where the number of bytes of status information available is to be returned.

**Request Block**

BmQuery uses the same request block as BmIdentify.

The format of the status block returned is:

| Offset | Field | Size (bytes) | Comments |
|---|---|---|---|
| 0 | Line Activity | 48 | consists of four buffers of 12 bytes which contain control strings seen by BMULTI. This contains typically a Poll string or Select string and could be used by the application to determine which addresses are actually polled by the host. The format of each buffer is shown below. |
| 48 | Station Block | 1 | For each active address BMULTI maintains a Station record whose size in bytes is indicated in this byte. |

| | | | |
|---|---|---|---|
| 49 | Active Stations | 32 | This is a boolean array where each byte indicates which Station Record is actually active and in use. This should be used along with the station block size to compute the offset of the station record. |
| 81 | Station Information | Var | Each station record contains several pieces of information, such as the device address which is currently using this record. Depending on the status buffer size provided by the application, integral number of station records are returned in this area. |

# HIGH-LEVEL INTERFACE PROCEDURES

## OpenBmulti

**Description**

This procedure opens a Bmulti station with the supplied device address and returns a task handle which should be used for all successive Bmulti calls. This procedure also creates the HLI process. The application must provide an Application Status Block (ASB) which is used by the HLI process to communicate to the application the status of the 'reads' and 'writes' it issues.

**Procedural Interface**

```
OpenBMULTI (devAdr, fSys, Priority, pTskH, pASBlk):
                                                Erctype
```

where

devAddr is a word containing two ASCII characters. This address must not duplicate that of any other stations on the same line.

fSys is a byte or Boolean. It should be set to TRUE if the application making the call is to be a system service.

Priority is a word, containing the priority with which the HLI process is to be created. (This is normally 128).

pTskH is a pointer to a byte

pASBlk is a pointer to the application-supplied Application Status Block (ASB). The format of the ASB is given below.

**Format of the Application Status Block:**

| Field | Size bytes | Appln Usage | Comment |
|---|---|---|---|
| RcvStatus | 1 | R | Used by HLI process to indicate 'Read' status. |
| RcvErc | 2 | R | Internal error seen by the HLI process for Receive. |
| fSelDen | 1 | R/W 0 | Set by HLI process to True if Line is selecting this address. |
| Xmtstatus | 1 | R | Used by HLI process to indicate 'Write' status. |
| Xmterc | 2 | R | Internal error seen by the HLI process for Xmt. |
| Option | 1 | R | station option byte (used by SetOptionBmulti). |
| fFMess | 1 | R/W 0 | Set to True if a Fast Select Message has been received by the HLI process for this address. |
| pFMess | 4 | R | Address of Fast Message Buffer (which is not in the application area). This pointer is returned by the HLI process after an 'OpenBmulti'. |

## ReadBmulti

**Description**

This procedure initiates a receive operation. It does not wait for the message actually to arrive. The application should sample the Rcvstatus of the ASB to determine when the receive is complete. When complete, the HLI process returns the received message in the receive buffer with the first two bytes containing the length of the message received.

**Procedural Interface**

```
ReadBMULTI (TskH, pBuf, sBuf) : Erctype
```

where

TskH is a byte (returned by OpenBmulti).

pBuf is a pointer to an application-supplied buffer where the received data is to be placed.

sBuf is a word containing the maximum number of bytes which the application can receive.

Possible values of RcvStatus (Hex ):

| | |
|---|---|
| 00 | Initial state or after a 'reset' |
| 01 | Busy (Read initiated) |
| 02 | Read failure. |
| 10 | Read complete (no errors) |
| 11 | Read complete (Transmission number error) |
| 12 | Read complete (Duplicate transmission number) |
| 14 | Read complete (Truncated message) |

## WriteBmulti

**Description**

This procedure initiates a transmit operation. It does not wait for the message to be successfully transmitted. The application should sample the Xmtstatus of ASB to determine when the tranmission is complete.

**Procedural Interface**

```
WriteBMULTI (TskH, pBuf, sBuf) : Erctype
```

where

TskH is a byte (returned by OpenBmulti).

pBuf is a pointer to an application-supplied buffer which contains the data Bmulti is to transmit.

sBuf is a word containing the number of bytes Bmulti is to transmit.

Possible values of XmtStatus (Hex):

| | |
|---|---|
| 00 | Initial state or after a 'reset' |
| 01 | Busy (Write initiated) |
| 02 | Write failure. |
| 10 | Write Complete |

## SetOptionBmulti

**Description**

This procedure enables the application to select certain Bmulti options.

**Procedural Interface**

```
SetOptionBMULTI (TskH, Option) : Erctype
```

where

TskH is a byte (returned by OpenBmulti).

Option is a byte with values as follows:

```
Bit 0    1 = Offline
         0 = Online

Bit 1    1 = Fast Rdy
         0 = Not Fast Rdy
```

The ASB option field is updated to reflect the option selected.

## ResetBmulti

**Description**

This procedure can be used to restore the application status to 'idle'. It can also be used to abort an ongoing read or write. After a Reset the option byte is set to 0.

**Procedural Interface**

```
ResetBMULTI (TskH) : Erctype
```

where

TskH is a byte (returned by OpenBmulti).

## Close Bmulti

**Description**

This procedure frees the device address by issuing an end session command. This is normally the last step in a datacomm session.

**Procedural Interface**

```
CloseBMULTI (TskH) : Erctype
```

where

TskH is a byte (returned by OpenBmulti).

**Table 3-5. Bmulti.lib Module Requirements and Procedure Calls**

| Interface Used | Bmulti.lib Module | Memory Required (bytes) | Procedures Served |
|---|---|---|---|
| Single-Task Interface | BMCmdRpt | 1700 | DCCOMMAND<br>DCCONFIG<br>DCREPORT<br>DCREPORTWAIT |
| Multiple-Task Interface | BMCmdRptMP | 2000 | MPCOMMAND<br>MPREPORT<br>MPREPORTWAIT |
| Multitasking Low-Level Interface | Bmx1 | 2900 | BMCOMMAND<br>BMOPEN<br>BMREPORT<br>BMREPORTWAIT |
| | BMX2 | 700 | BMIDENTIFY<br>BMQUERY |
| Multitasking High-Level Interface | Hli5 | 7000* | CLOSEBMULTI<br>OPENBMULTI<br>READBMULTI<br>RESETBMULTI<br>SETOPTIONBMULTI |

* The object module Hli5 uses the module Bmx1; therefore the memory required when using the Multitasking High-Level Interface is actually the combination of the memory required for each object module, or 9900 bytes.

# SECTION 4
# PROTOCOL DESCRIPTION

## GENERAL

This section provides a description of the Burroughs multipoint protocol. As implemented in the B20, this includes only the terminal side of the protocol (that is, the B20 acts like a Burroughs terminal).

The diagrams include all of the protocol options implemented in BMULTI. Most users will use only a few of these.

For asynchronous data communication, each transmitted character utilizes ten nominally equal time intervals. The time intervals represent a start bit, 8 bits of information, and a stop bit. Of the 8 information bits, 7 represent an ASCII character and the eighth is a parity bit selected to make the number of 1, or marking bits of the 8 bit group even.

For synchronous data communications, each transmitted character utilizes eight nominally equal time intervals, representing 8 bits of information. The first 7 bits represent the 7 bit character code, transmitted with the least significant bit first. The eighth bit is to be a parity bit selected to make the number of 1, or marking bits of the 8 bit group odd. The following transmission character follows immediately, with no inter-character interval.

## STATION TYPES

Contained within this section are references to control and terminal stations. The following definitions are to be used in understanding these references. A control station is that station on a data link with the overall responsibility for polling, selecting, and otherwise ensuring the orderly operation of that link. (Usually a control station is a large Burroughs computer.) Responsibility to initiate recovery procedures in the event of abnormal conditions on the link rests with the control station. All stations on a multipoint network, other than the control station, are called terminal stations. These are usually terminals, but may be microcomputers (as in the case of the B20) or even minicomputers.

## Control Characters

The following is a list of control characters and an explanation of each.

ACK (ACKNOWLEDGEMENT, 06h)
This is an affirmative response to a normal selection (indicating Ready to Receive) or a transmission (indicating Message Accepted).

BCC (BLOCK CHECK CHARACTER)
This is a redundant character added to the end of a message for the purpose of error detection and control. BCC is formed by taking a binary sum without carry on each of the 7 bits of the transmitted characters following SOH, including ETX, but excluding any SYN characters. The correct value of the character parity bit of the BCC is that which makes the sense of character parity the same as for text characters. BCC immediately follows ETX.

ENQ (INQUIRY, 05h)
This is a reply request control character. The ENQ is used as the final character of a poll or of a select, when a response is required from the other station.

EOT (END OF TRANSMISSION, 04h)
EOT is transmitted by a terminal as a No Traffic response to a poll. Receipt of EOT places the terminal in a control state listening for a polling or selection sequence. EOT may be transmitted instead of ETX to abort a transmission.

ETX (END OF TEXT, 03h)
This is used to indicate the end of a stream of characters identified as a text.

NAK (NEGATIVE ACKNOWLEDGEMENT, 15h)
This is a negative response to a selection (indicating Not Ready to Receive ) or a transmission (indicating character parity failure for any character in a message or a failure of the BCC).

RVI (REVERSE INTERRUPT (DLE <, 103Ch)

Reverse Interrupt is sent by the control station in lieu of a positive acknowledgement (ACK) when the control station has priority messages to deliver. RVI is normally used in a group poll environment to request premature termination of a series of message transmissions, in order to allow the control station to either transmit return messages or to poll other terminals. Upon receipt of an RVI, the terminal should send EOT as soon as possible.

SOH (START OF HEADING, 01h)

SOH is the first of a sequence of characters which form the heading. The heading also contains a terminal identification (AD1, AD2) and may contain transmission numbers (XM#). A heading is ended by STX.

STX (START OF TEXT, 02h)

This precedes a sequence of characters which form the text of the transmission. STX terminates a heading.

SYN (SYNCHRONOUS IDLE, 16h)

This is used only with synchronous transmission in the absence of any other character to provide a signal for establishing and retaining synchronism. On initiating a synchronous transmission, a number of SYN characters are transmitted prior to the transmission of any character. This permits the receiving station to acquire character synchronization. SYN is also used as a time fill when no other characters are available for transmission at any point in a character sequence, except between ETX and the next following BCC. SYN is purged at the receiving station and is not included in the summation for BCC.

## Additional Procedural Characters.

The following characters are additional procedural characters which may have significance outside of the multipoint protocol.

AD1, AD2 (ADDRESS 1, ADDRESS 2)

This is a two character address established as the address of a terminal. These characters are used to address a terminal in polling or selection or in the message heading. These characters identify the terminal from which a message is transmitted. On receipt of a message, the receiving station may use AD1 - AD2 to verify that the message originated at the polled terminal. AD1 and AD2 are represented by any characters from columns 2,3,4,5,6,7 of the ASCII code chart on page B-2, except the character DEL, column 7, row 15, shown as 7/15.

BSL (BROADCAST SELECT, 74h)

This is a character used to indicate a broadcast message to all stations. In the broadcast sequence, AD1 - AD2 identifies the station which acknowledges receipt of the message. Broadcast select is followed immediately by a message without requiring acknowledgement of the selection.

CON (CONTENTION, 07h)

This is a character used to instruct all terminals which receive the instruction to go to the contention mode. NUL characters replace AD1 - AD2 in the contention sequence. There is no acknowledgement of the contention instruction.

FSL (FAST SELECT, 73h)

This is a character used to indicate a Fast Select, in a selection sequence transmitted by the central computer. Fast Select is followed immediately by a message without requiring acknowledgement of the selection.

GSL (GROUP SELECT)

This is a character used to indicate a Message for a Group of Stations. In the group select sequence, AD1 - AD2 identifies the station which is to acknowledge receipt of the message. Group select is followed immediately by a message without requiring acknowledgement of the selection. Group select may be represented by any agreed on character selected from column 2 through 6.

POL (POLL, 70h)

This is a character used to indicate a Poll, preceding ENQ in a polling sequence.

SEL (SELECT, 71h)

This is a character used to indicate a Normal Select, preceding ENQ in a selection sequence.

XMno (TRANSMISSION NUMBER)

This is a number identifying, in sequence, transmissions from or a transmission to a terminal. It is optionally used as part of a message header to assist in message recovery. Separate sets of transmission numbers are to be used for broadcast and group addressed messages.

In transferring data from one point to another, proper accountability for each message is required under certain conditions. For example, in the handling of financial transactions, such as electronic transfer of funds where large amounts of money are transferred between banks via telecommunications, it is imperative that messages are not lost, and that they are not duplicated. Where loss of a message or duplication is not important, transmission numbering may not be considered necessary. If each message sent has a transmission number serially assigned to it, the receiver can check for the following.

a. Each message sent is received.

b. Messages are received in the order sent.

c. A message is not a retransmission of a previously transmitted message; therefore, it is not handled twice.

It should be noted that the message numbers sent from one end need have no relationship to those sent from the other end. Data transmission is not a balanced function; that is, one message sent does not always result in one reply.

ALTERNATING TRANSMISSION NUMBERING

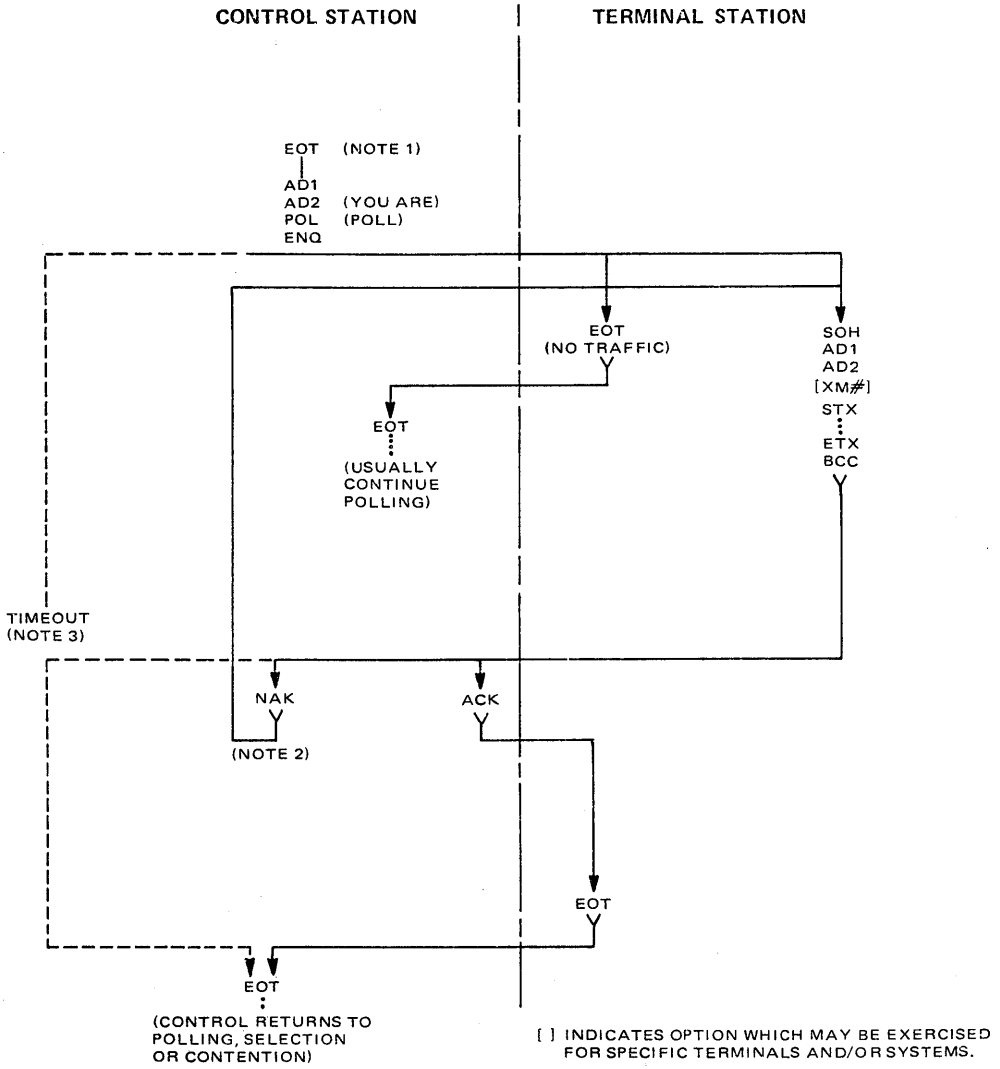
The minimum level of message numbering is a single character that alternates between an even and odd state. This system cannot distinguish between an error caused by loss of a message and one caused by duplication of a message, though normal protocol procedures should prevent loss of a message. The B20 allows for alternating a 0 and 1, or for alternating an @ and A.

SEQUENTIAL TRANSMISSION NUMBERING
This provides more positive indication of the loss or duplication of messages than the odd/even method. The B20 allows one, two, or three digit transmission numbers starting at 0 (or 00 or 000) and cycling through, respectively, 9, 99, and 999.

**Time-outs**

NO RESPONSE TIMEOUT.
The timing starts after transmission of a character signifying reversal of transmission direction. The time is to range from 1 to 3 seconds. If the first character of a terminal transmission is not received, or if the character received is not valid in its time, the controller or terminal repeats its transmission 'n' times ('n' $\leq$ zero), and then, if the same condition exists, it interrupts and enters the necessary error recovery procedures. If the reversal is a result of an ACK or NAK, no repeat of the ACK or NAK is sent, but EOT is sent to return to the control state.

IDLE LINE.
The timing starts on receipt of each character other than a character signifying reversal of direction of transmission. Time is to range from 1 to 3 seconds. If the next character is not received in this time, the central processor interrupts and enters the necessary error recovery procedures.




NOTES:

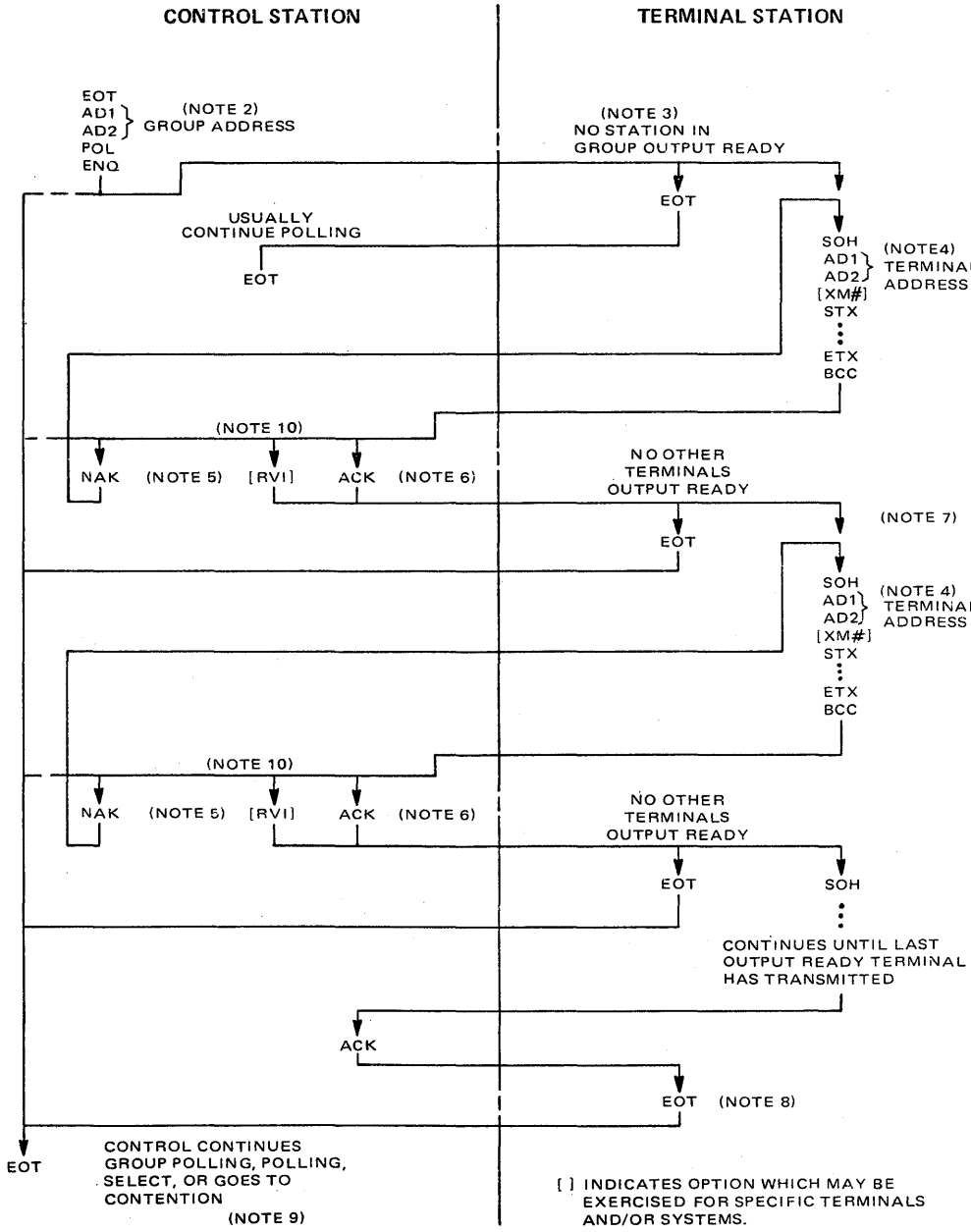
1. This EOT must come from the control station and may have been the termination of a previous transmission sequence. To minimize the effect of noise, the polling sequence may follow immediately.

2. If the control station receives a message for which character parity or block check test fails, NAK is transmitted, calling for a repeat of the transmission. This can be repeated 'n' times (to be defined by the control station programmers), at which time, if the test fails, an error is recorded at the control station and EOT is transmitted, terminating the sequence. The terminal transmits the same message when next polled.

3. If the terminal does not receive ACK, NAK, or EOT, it may retain its message and remain quiet. The control station time outs and transmits EOT, terminating the sequence. In this case the message is retransmitted when next polled.

E4336

**Figure 4-1. Specific Polling**




NOTE: SEE SHEET 2 FOR NOTES.

E4337A

**Figure 4-2. Group Polling (Note 1), Sheet 1**

NOTES:

1. This procedure is used to reduce the overhead in a network of terminals where several are located at one location on a common communication line. The receipt of one group poll results in one response for the group if no terminals are output ready. Thus the control station can pass to the next group. In periods of low activity, the control station has the ability to go through the polling list determining the output status of all terminals with but one poll to each location, not each terminal. Also, if multiple terminals are output ready at a location, they are allowed to transmit, in sequence, in response to one poll. Selecting, broadcast select, fast select, etc., are not affected by this group polling procedure. This procedure may also be used with B20s when several applications running on one B20 cluster system are using BMULTI.

2. In this procedure the polling sequence follows the same format as a normal poll and uses the normal poll character. Group polling is controlled by address only. Terminals at a common location that are to be a part of a group are so identified by making their group poll addresses all the same. All application programs using BMULTI in a B20 system have the same group poll address.

3. When the poll is received by the group addressed, the output ready terminals respond in the normal manner.

4. Each message sent in response to a group poll contains the address of the individual terminal which is responding.

5. If the control station detects an error in the message received in response to a group poll, normal polling error recovery is used.

6. The control station must, under this procedure, be sure when it replies ACK to a message that buffer space exists or is to be available for the next message that could result from another output ready terminal.

7. As soon as ACK is received from the control station, the next output ready terminal transmits.

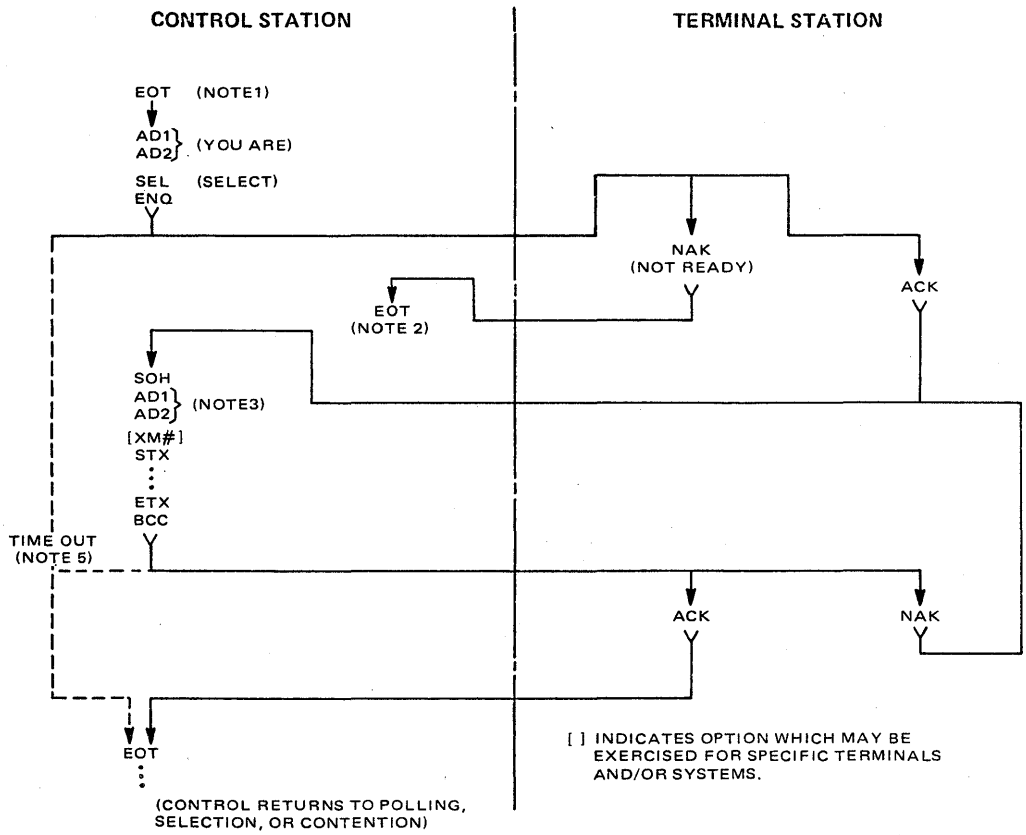
8. When an ACK is received from the control station and no terminals remain output ready, the last terminal on the line must transmit the final EOT.

9. The same error recovery procedure as is outlined for figure 4-1 is used with this procedure.

10. Reverse interrupt (RVI) may be used by the control station only after reception of a valid message which would result in a positive acknowledgement. In place of sending ACK the control station sends RVI (DLE < ). When RVI is received from the control station, the last terminal to transmit sends the final EOT even if other terminals are still output ready. In this case the other output ready terminals retain their messages until the next group poll (or normal poll to their address).

E4337B

**Figure 4-2. Group Polling (Note 1), Sheet 2**

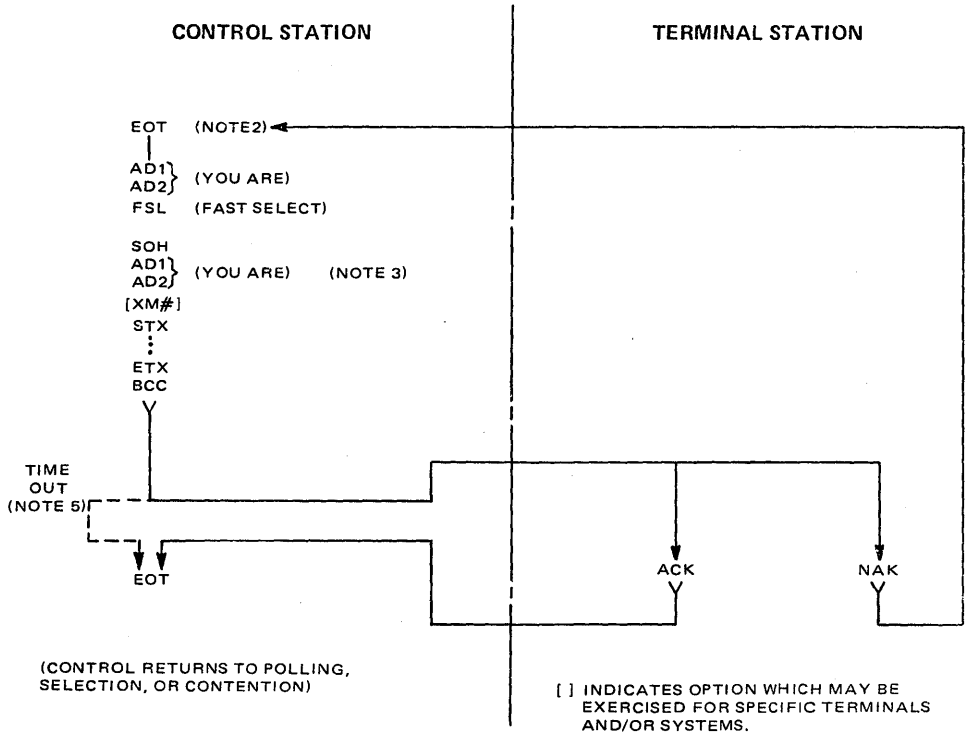



NOTES:

1. This EOT must come from the control station and may have been the termination of a previous transmission sequence. To minimize the effect of noise, the selection sequence may follow immediately.
2. If the terminal is not ready to receive, as indicated by transmission of NAK, the control station normally retries the selection at the proper sequence of that terminal.
3. The identification characters in a transmission represent the terminal address for selection verification purposes. If the terminal fails to verify the address, it ignores the message.
4. If character parity or block check are not validated by the terminal, it sends NAK. In this case the control station retransmits the message 'n' times ('n' may be equal to zero). If the terminal still does not acknowledge the message, the control station terminates the sequence with EOT, after recording the error. The control station retains the message for transmission on the next selection sequence to this terminal.
5. If the control station does not receive a response (ACK or NAK) to its message, it may time out and retransmit the message 'n' times ('n' may equal zero). If still no response is received, the control station terminates the sequence with EOT, after recording the error. The control station retains the message for transmission on the next selection sequence to this terminal.

E4338

**Figure 4-3. Selection**




NOTES:

1. Fast selection is used when the control station wishes to send a message to a terminal without first testing to make sure that the terminal is ready to receive. In this case, the selection and the message are transmitted together. The ACK response from the terminal applies to both the select and to a successful message transfer. A NAK response may indicate either that the terminal is not ready to receive or that the parity or block check in the message is invalid.

2. This EOT must come from the control station and may have been the termination of a previous transmission sequence. To minimize the effect of noise, the fast selection sequence may follow immediately.

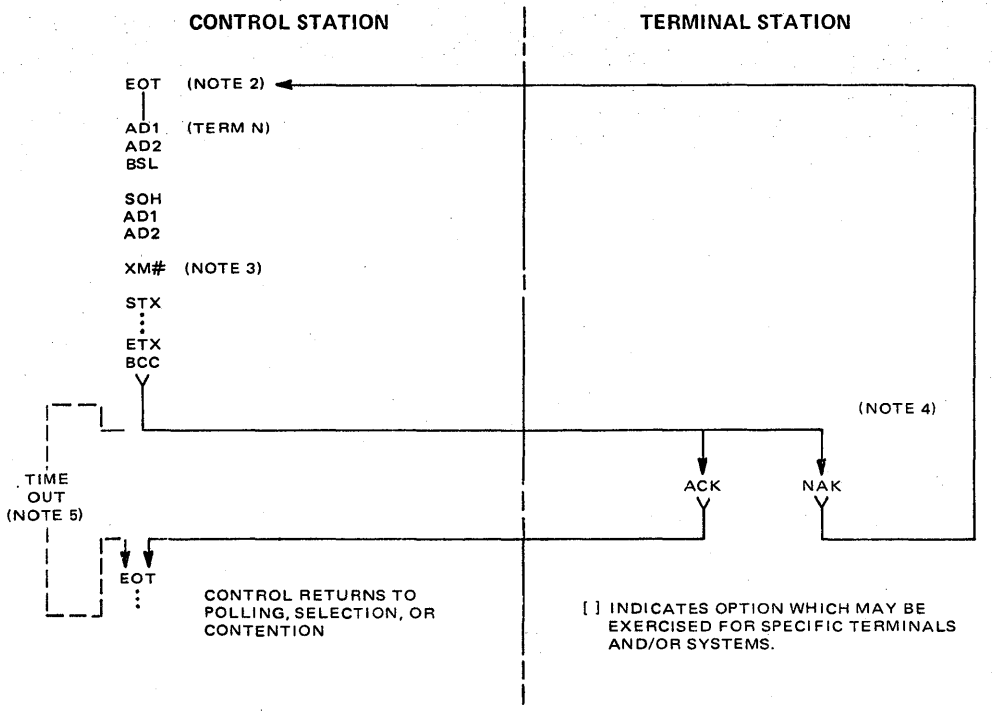
3. The identification characters in a transmission from the control station also represent the terminal address YOU ARE for selection verification purposes. If either pair of addresses fail to verify, the terminal ignores the message.

4. If character parity or block check are not validated by the terminal selected, or if the terminal selected is not ready to receive the message, it responds NAK. In this case, the control station resends the fast select transmission 'n' times ('n' may equal zero). If the terminal still does not accept the message, the control staion terminates the sequence and retains the message for transmission on the next selection sequence to this terminal.

5. If the control station does not receive a response (ACK or NAK) to its transmission, it time outs and terminates the sequence. The control station retains the message for transmission on the next normal selection for this terminal.

E4339

**Figure 4-4. Fast Select (Note 1)**

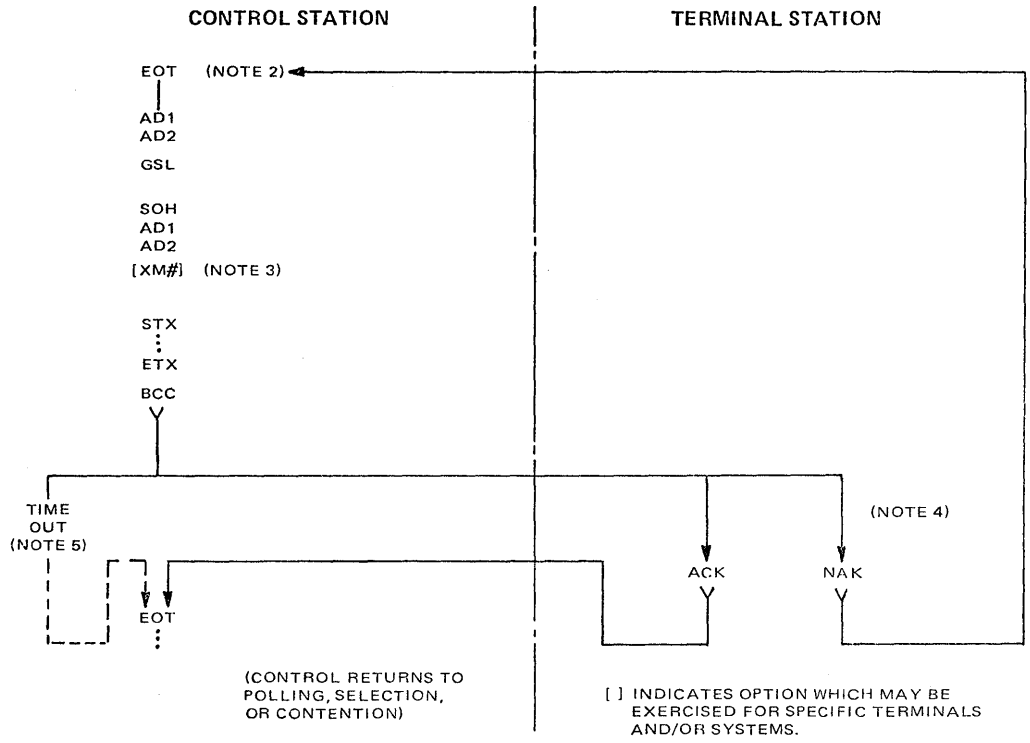



NOTES :

1. Broadcast select is a fast selection of all terminals. AD1-AD2 is selected to represent the terminal which acknowledges receipt of the message.
2. This EOT must come from the control station and may have been the termination of a previous transmission sequence. To minimize the effect of noise, the broadcast sequence may follow immediately.
3. Special sequences of numbers must be maintained if transmissions are numbered in a system where broadcast is employed.
4. If the acknowledging terminal does not receive a valid message (e.g., there is a character parity or block check error) or is not receive ready, it transmits NAK. The control station has the option of repeating the total broadcast.
5. If the control station does not receive a response (ACK or NAK) to its broadcast, it may time out and rebroadcast the message 'n' times ('n' may equal zero). If no response is received, the control station terminates the broadcast mode with EOT after recording the error.

E4340

**Figure 4-5. Broadcast Select (Note 1)**

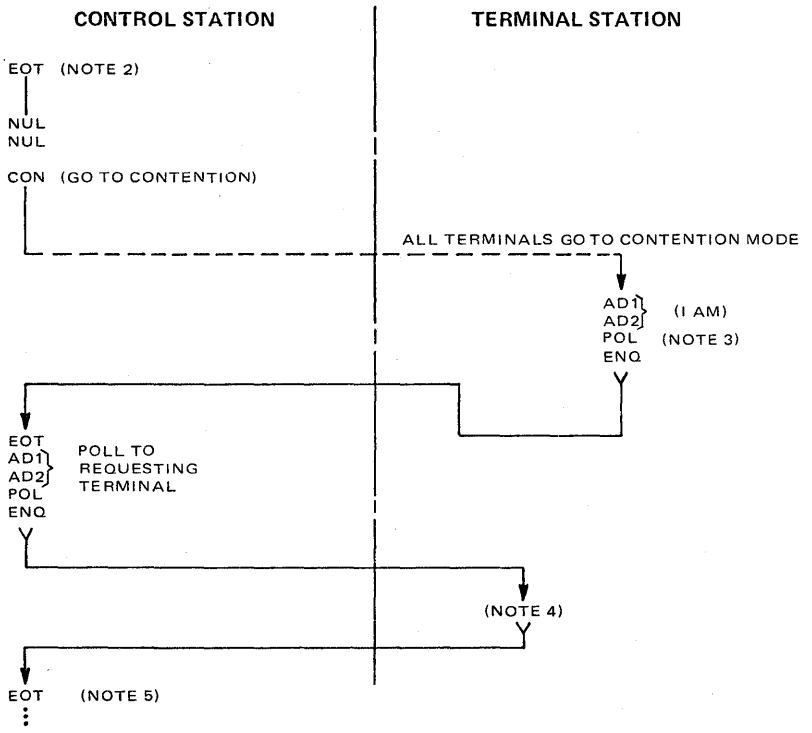



NOTES:

1. Group selection is a fast selection of a group of terminals. Each terminal may have a group select character for which it accepts a message. AD1-AD2 is selected to represent the address of the terminal which acknowledges receipt of the message.
2. This EOT must come from the control station and may have been the termination of a previous transmission sequence. To minimize the effect of noise, the group selection may follow immediately.
3. Special sequences of numbers must be maintained if transmissions are numbered in a system where group selection is employed.
4. If the acknowledging terminal does not receive a valid message (e.g., there is character parity or block check error) or it is not receive ready, it transmits NAK. The control station has the option of repeating the total group selection.
5. If the control station does not receive a response (ACK or NAK) to the group selection, it may time out and reselect the group 'n' times ('n.' may equal zero). If no response is received, the control station terminates the group select mode with EOT after recording the error.

E4341

**Figure 4-6. Group Select (Note 1)**




NOTES:

1. In times of low activity, it may be desirable to terminate polling and to place all or part of the system in the contention mode. This is accomplished by transmission of EOT NUL NUL CON which causes the terminals to remain quiet until they have something to transmit.

2. This EOT must come from the control station and may have been the termination of a previous transmission sequence. To minimize the effect of noise, the Go to Contention sequence may follow immediately.

3. A terminal may Wake Up the polling activity by transmitting AD1 AD2 POL ENQ. This causes the control station to poll that terminal. If two terminals attempt to transmit at the same time, the garbled message initiates general polling by the control station.

4. The terminal proceeds with normal message transfer as in response to a poll (see figure 4-1).

5. Following normal message receipt verification procedures, as in figure 4-1, the control station may continue polling or instruct all terminals to Go to Contention.

E4342

**Figure 4-7. Multipoint Contention Mode (Note 1)**

# APPENDIX A
# SAMPLE PROGRAMS

## GENERAL

In order to illustrate the programming technique used in writing an application to interface with BMULTI, sample programs are reproduced below in FORTRAN, COBOL, BASIC and Pascal.

NOTE: FORTRAN, BASIC, and COBOL languages require reconfiguration before such applications can be run. See Appendix D for details of language configuration. All languages require that Bmulti.lib be linked into the executable code file.

The sample programs echo any text received back to the host system, at any buffer length up to the maximum allowed.

## PASCAL ECHO PROGRAM

```
{$DEBUG-}
{$ENTRY-}

PROGRAM echo(INPUT,OUTPUT);

TYPE
  String2      = STRING(2);
  pointr       = ADS OF BYTE;

CONST

    (* Command codes for BMULTI *)
  Txrebuf           = 16#0001;
  Txtxbuf           = 16#0002;
  Offlinec          = 16#0003;
  Onlinec           = 16#0004;
  Idlec             = 16#0005;
  Fastsetc          = 16#0006;
  Receivec          = 16#0007;
  Transmitc         = 16#0008;
  Endsessionc       = 16#0009;
  Abortc            = 16#000A;
  Fastresetc        = 16#000B;

    (* Error return codes for BMULTI *)
  System_error      = 16#0000;
  Command_Accepted  = 16#0001;
  Command_Denied    = 16#0002;

    (* Report codes for BMULTI *)
```

```
  No_report          = 16#0000;
  Receiving          = 16#0001;
  Rec_Fast_Sel       = 16#0002;
  Receive_Done       = 16#0003;
  Rdy_Xmt_Xfer       = 16#0004;
  Transmit_Done      = 16#0005;
  Select_Denied      = 16#0006;
  Receive_err        = 16#0007;
  Dup_seq_num        = 16#0008;
  Seq_num_err        = 16#0009;
  Transmit_err       = 16#000A;
  Internal_err       = 16#00FF;

VAR
  Erc          : INTEGER;
  report       : INTEGER;
  etx          : CHAR;
  done         : BOOLEAN;
  cntr1        : INTEGER;
  cntr2        : INTEGER;
  devadr       : STRING(2);
  Buff         : ARRAY[0..2047] of CHAR;
  sBuf         : INTEGER;
  sBufMax      : INTEGER;

VALUE
  etx            := CHR(03);
  sBufMax        := 2048;

FUNCTION  DcConfig (addr : String2) : INTEGER; EXTERN;
FUNCTION  DcReport : INTEGER; EXTERN;
FUNCTION  DcCommand (comm    : WORD;
                     pbuffer : pointr;
                     sbuffer : INTEGER) : INTEGER; EXTERN;
PROCEDURE Exit; EXTERN;

PROCEDURE Error;
BEGIN
  WRITELN('Command Denied', Erc);
  Exit;
END;

BEGIN
  WRITELN('BMULTI echo program ');

  REPEAT

    WRITELN('Enter Station Address: ');
    READLN(devadr);
    Erc := DcConfig (devadr);
    IF (Erc = System_error) THEN Error;
  UNTIL (Erc <> Command_Denied);

  WRITELN('Begin BMULTI');
```

```
  Erc := DcCommand (Onlinec, ADS buff [Ø], Ø);
  IF (Erc <> Command_Accepted) THEN Error;

  FOR cntr1 := 1 TO 5Ø DO
    BEGIN

      FOR cntr2 := 1 TO sBufMax DO
        BEGIN
          buff [cntr2] := ' ';
        END;

      Erc := DcCommand (Receivec, ADS buff [Ø], Ø);
      IF (Erc <> Command_Accepted) THEN Error;

      REPEAT report := DcReport;
      UNTIL  (report = Receive_Done)
          OR (report = Seq_Num_Err);

      Erc := DcCommand (Txrebuf, ADS buff [Ø], sBufMax);
      IF (Erc <> Command_Accepted) THEN Error;

      sBuf := Ø;
      WHILE (buff [sBuf] <> etx) AND (sBuf < sBufMax) DO
          sBuf := sBuf + 1;

      Erc := DcCommand (Transmitc, ADS buff [Ø], Ø);
      IF (Erc <> Command_Accepted) THEN Error;

      REPEAT report := DcReport;
      UNTIL  (report = Rdy_Xmt_Xfer);

      Erc := DcCommand (Txtxbuf, ADS buff [Ø], sBuf);
      IF (Erc <> Command_Accepted) THEN Error;

      REPEAT report := DcReport;
      UNTIL  (report = Transmit_Done);
    END;

  WRITELN('End BMULTI Echo Program');
  REPEAT Erc := DcCommand (Endsessionc, ADS buff [Ø], Ø);
  UNTIL  (Erc = Command_Accepted);

  Exit;
END.
```

# COBOL ECHO PROGRAM

NOTE: In this COBOL program, the station address must be entered backwards, e.g. if the address is to be "A1", then "1A" must be entered.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  Echo.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  B20.
OBJECT-COMPUTER.  B20.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
77  counter          PIC 9(4) COMP.
01  miscellaneous.
    03  device-address    PIC XX.
    03  ercdisplay        PIC 9(4).
    03  buffer-size       PIC 9(4) COMP.
    03  max-buffer-size   PIC 9(4) COMP  VALUE 2048.
    03  nil               PIC 9(4) COMP  VALUE 0.
    03  etx               PIC X          VALUE X"03".
*
01  buffer-whole.
    03  buffer       PIC X(2048).
01  buffer-array    REDEFINES buffer-whole.
    03  byte         PIC X(1)       OCCURS 2048.
*
01  Command-codes.
    03  Transfer-Recv-buf     PIC 9(4) COMP VALUE 1.
    03  Transfer-Xmit-buf     PIC 9(4) COMP VALUE 2.
    03  offline               PIC 9(4) COMP VALUE 3.
    03  online                PIC 9(4) COMP VALUE 4.
    03  idle                  PIC 9(4) COMP VALUE 5.
    03  set-fast-ready        PIC 9(4) COMP VALUE 6.
    03  receive-ready         PIC 9(4) COMP VALUE 7.
    03  transmit-ready        PIC 9(4) COMP VALUE 8.
    03  end-session           PIC 9(4) COMP VALUE 9.
    03  Abort                 PIC 9(4) COMP VALUE 10.
    03  reset-fast-ready      PIC 9(4) COMP VALUE 11.
*
01  Error-code           PIC 9(4) COMP.
    88  System-error              VALUE 0.
    88  No-Error                  VALUE 1.
    88  Address-Is-Good           VALUE 1.
    88  Command-denied            VALUE 2.
*
```

```
 Ø1  Report                PIC 9(4) COMP.
     88  No-Report                  VALUE Ø.
     88  Receiving                  VALUE 1.
     88  Receiving-Fast-Select      VALUE 2.
     88  Receive-Done               VALUE 3.
     88  Ready-for-Xmit-Buf         VALUE 4.
     88  Transmit-Done              VALUE 5.
     88  Select-Denied              VALUE 6.
     88  Receive-Error              VALUE 7.
     88  Dup-Seq-Num                VALUE 8.
     88  Seq-Num-Err                VALUE 9.
     88  Transmit-Error             VALUE 1Ø.
     88  Internal-Error             VALUE 255.
*
 PROCEDURE DIVISION.
 MAIN-LINE.
     PERFORM start-up.
     PERFORM driver THRU driver-x
         VARYING counter FROM 1 BY 1
             UNTIL counter IS EQUAL TO 5Ø.
     PERFORM finish-up.
*
 START-UP.
     DISPLAY "BMULTI echo program" UPON CONSOLE.
     PERFORM Get-Address THRU Get-Address-X
         UNTIL Address-Is-Good.
     DISPLAY "Begin BMULTI" UPON CONSOLE.
     CALL "&DCCOMMAND" USING error-code, online,
                             buffer, nil.
     PERFORM error-check.
*
 DRIVER.
     MOVE SPACES TO buffer.
     CALL "&DCCOMMAND" USING error-code, receive-ready,
                             buffer, nil.
     PERFORM error-check.
     PERFORM reporting UNTIL Receive-Done OR Seq-Num-Err.
     CALL "&DCCOMMAND" USING error-code,
                             Transfer-Recv-buf, buffer,
                             max-buffer-size.
     PERFORM error-check.
     MOVE 1 TO buffer-size.
     PERFORM add-1-to-buffer-size
             UNTIL byte(buffer-size) IS EQUAL TO etx
               OR buffer-size IS EQUAL TO max-buffer-size.
     SUBTRACT 1 FROM buffer-size.
     CALL "&DCCOMMAND" USING error-code, transmit-ready,
                             buffer, nil.
     PERFORM error-check.
     PERFORM reporting UNTIL Ready-for-Xmit-Buf.
     CALL "&DCCOMMAND" USING error-code,
                             Transfer-Xmit-buf, buffer,
                             buffer-size.
     PERFORM error-check.
     PERFORM reporting UNTIL Transmit-Done.
```

```
*
 DRIVER-X.
     EXIT.
*
 FINISH-UP.
     CALL "&DCCOMMAND" USING error-code, end-session,
                             buffer, nil.
     IF Command-Denied THEN
         GO TO finish-up.
     STOP RUN.
*
 GET-ADDRESS.
     DISPLAY "Enter Station Address: " UPON CONSOLE.
     ACCEPT device-address.
     CALL "&DCCONFIG" USING error-code, device-address.
     IF System-Error THEN
         PERFORM print-error.
 GET-ADDRESS-X.
     EXIT.
*
 REPORTING.
     CALL "&DCREPORT" USING Report.
*
 ADD-1-TO-BUFFER-SIZE.
     ADD 1 TO buffer-size.
*
 ERROR-CHECK.
     IF System-Error OR Command-Denied THEN
         PERFORM print-error.
*
 PRINT-ERROR.
     MOVE error-code TO ercdisplay.
     DISPLAY "Command Denied ", ercdisplay UPON CONSOLE.
     STOP RUN.
*
 END-OF-JOB.
```

# FORTRAN ECHO PROGRAM

```
$STORAGE: 2

      SUBROUTINE Error(int)
        CHARACTER*20 msg
        DATA msg/' Command Denied'/
        WRITE (*,'(A,14)') msg, int
        CALL Tmexit
      END

      SUBROUTINE ChkErc(int)
        INTEGER*2 ercOk
        DATA ercOk/1/
        IF (int.EQ.ercOk) RETURN
        CALL Error(int)
      END

      PROGRAM echo
        IMPLICIT INTEGER*2 (d)
        EXTERNAL DcCmnd, DcRept, DcCnfg
        INTEGER*2 erc, report, sBuf, sBufMx, xfrrcv,
     +            xfrxmt, ofline, online, idle, fstrdy,
     +            rcvrdy, xmtrdy, endit, abortc, fstrst,
     +            sysErr, ercOk, comNak,noRept, rcving,
     +            rcvGrp, rcvFin, rdyXfr, xmtFin, selNak,
     +            rcvErr, dupSeq, seqErr, xmtErr, i

        CHARACTER*1 key, etx, Buf
        CHARACTER*2 devadr
        CHARACTER*24 msg
        DIMENSION msg(5), Buf(1024)

        DATA  sBufMx/2048/
c
c  command codes for Bmulti
c
        DATA xfrrcv/1/,
     +       xfrxmt/2/,
     +       ofline/3/,
     +       online/4/,
     +       idle/5/,
     +       fstrdy/6/
        DATA rcvrdy/7/,
     +       xmtrdy/8/,
     +       endit/9/,
     +       abortc/10/,
     +       fstrst/11/
```

```
c
c  Error return codes for Bmulti
c
        DATA sysErr/0/,
     +       ercOk/1/,
     +       comNak/2/
c
c  Report codes for Bmulti
c
        DATA noRept/0/,
     +       rcving/1/,
     +       rcvGrp/2/,
     +       rcvFin/3/,
     +       rdyXfr/4/,
     +       xmtFin/5/
        DATA selNak/6/,
     +       rcvErr/7/,
     +       dupSeq/8/,
     +       seqErr/9/,
     +       xmtErr/10/

        DATA msg(1)/' BMULTI echo program'/,
     +       msg(2)/' Enter Station Address: '/,
     +       msg(3)/' Begin BMULTI'/,
     +       msg(4)/' End BMULTI echo program'/
c
c
c  Beginning of procedural code
c
c
         etx = CHAR(3)
         WRITE (*,'(A)') msg(1)

 1       WRITE (*,'(A)') msg(2)
         READ  (*,'(A2)') devadr
         erc = DcCnfg (devadr)
         IF (erc.EQ.sysErr) CALL Error (erc)
         IF (erc.EQ.comNak) GOTO 1

         WRITE (*,'(A)') msg(3)

         CALL ChkErc(DcCmnd(Online, Buf(1), 0))

         DO 10 i = 1, 50

            CALL ChkErc(DcCmnd(Rcvrdy, Buf(1), 0))
```

```
2           report = DcRept()
            IF (report.NE.rcvFin) THEN
               IF (report.NE.seqErr) THEN
                  GOTO 2
               ENDIF
            ENDIF
            CALL ChkErc(DcCmnd(Xfrrcv, Buf(1), sBufMx))

            sBuf = 1
3           IF (Buf(sBuf).NE.etx) THEN
               IF (sBuf.LT.sBufMx) THEN
                   sBuf = sBuf + 1
                   GOTO 3
               ENDIF
            ENDIF

            CALL ChkErc(DcCmnd(xmtrdy, Buf(1), 0))

4           report = DcRept()
            IF (report.NE.rdyXfr) GOTO 4

            CALL ChkErc(DcCmnd(xfrxmt, Buf(1), sBuf-1))

5           report = DcRept()
            IF (report.NE.xmtFin) GOTO 5

10      CONTINUE

        WRITE (*,'(A)') msg(4)
9999    erc = DcCmnd(Endit, Buf(1), 0)
        IF (erc.NE.ercOk) GOTO 9999

        CALL Tmexit
      END
```

# BASIC ECHO PROGRAM

```
5  '******************************
7  '**                           **
10 '**    BASIC echo program     **
15 '**                           **
20 '******************************
112 DIM Msg$ [4]
114 DIM Buf% [1024]
118 i% = 0
120 Erc% = 0
122 Report% = 0
124 Daddr% = 0
130 DevAdr$ = "aa"
132 sBuf% = 0
134 sBufMax% = 2048
136 zero% = 0
140 etx% = 3
142 ibufsa% = 0
144 ibufra% = 0
145 incrra% = 0
146 '
147 '********************************
148 '**  Command codes for BMULTI  **
149 '********************************
150 xfrrcv% = 1
160 xfrxmt% = 2
170 ofline% = 3
180 online% = 4
190 idle%   = 5
200 fstrdy% = 6
210 rcvrdy% = 7
220 xmtrdy% = 8
230 endit%  = 9
240 abortc% = 10
242 fstrst% = 11
243 '
244 '************************************
245 '**  Error return codes for BMULTI  **
246 '************************************
250 System.Error% = 0
252 ErcOk%  = 1
254 Command.Denied% = 2
255 '
```

```
259 '*******************************
260 '**  Report codes for BMULTI  **
261 '*******************************
262 No.Report%       = 0
264 Receiving%      = 1
266 RecvGrpSel%     = 2
268 Receive.Done%   = 3
270 Ready.Xmit.Buf% = 4
272 Transmit.Done%  = 5
274 Select.Denied%  = 6
276 Receive.Error%  = 7
278 Dup.Seq.Err%    = 8
280 Seq.Num.Err%    = 9
282 Transmit.Error% = 10
340 '
342 Msg$ [1] = " BMULTI echo program"
350 Msg$ [2] = " Enter Address: "
360 Msg$ [3] = " Begin BMULTI"
370 Msg$ [4] = " End BMULTI Echo Program"
380 messag$  = " Command Denied"
390 '
392 '****************************************
394 '**    Beginning of procedural code    **
396 '****************************************
398 '
400 PRINT Msg$ [1]
402 '
405 PRINT Msg$ [2]
410 INPUT DevAdr$
415 Daddr% = CVI (DevAdr$)
420 Erc% = DcConfig (Daddr%)
422 IF Erc% = System.Error% THEN GOTO 1500
424 IF Erc% = Command.Denied% THEN GOTO 405
426 '
430 PRINT Msg$ [3]
435 '
440 Erc% = DcCommand(online%, PTR (Buf% [1]), zero%)
442 IF Erc% <> ErcOk% THEN GOTO 1500
444 '
510 ibufsa% = GETSA (PTR (Buf% [1]))
520 ibufra% = GETRA (PTR (Buf% [1]))
530 '*****************
540 '**  main loop  **
550 '*****************
560 FOR i% = 1 TO 50
570 '
600 Erc% = DcCommand (rcvrdy%, PTR (Buf% [1]), zero%)
602 IF Erc% <> ErcOk% THEN GOTO 1500
604 '
610 Report% = DcReportWait()
620 IF Report% = Receive.Done% THEN GOTO 660
630 IF Report% = Seq.Num.Err% THEN GOTO 660
650 GOTO 610
660 '
```

```
700 Erc% = DcCommand (xfrrcv%, PTR (Buf% [1]), sBufMax%)
702 IF Erc% <> ErcOk% THEN GOTO 1500
704 '
780 sBuf% = 0
790 incrra% = ibufra%
800 IF PEEK("B", MAKEPOINTER(incrra%, ibufsa%)) = etx% THEN
    GOTO 910
840 IF sBuf% >= sBufMax% THEN GOTO 910
850 sBuf% = sBuf% + 1
860 incrra% = incrra% + 1
870 GOTO 800
910 '
920 Erc% = DcCommand (xmtrdy%, PTR (Buf% [1]), zero%)
922 IF Erc% <> ErcOk% THEN GOTO 1500
924 '
930 Report% = DcReportWait()
970 IF Report% <> Ready.Xmit.Buf% THEN GOTO 930
990 '
1020 Erc% = DcCommand (xfrxmt%, PTR (Buf% [1]), sBuf%)
1022 IF Erc% <> ErcOk% THEN GOTO 1500
1024 '
1030 Report% = DcReportWait()
1070 IF Report% <> Transmit.Done% THEN GOTO 1030
1111 '
1120 NEXT i%
1122 '************************
1124 '**  end of main loop  **
1126 '************************
1130 PRINT Msg$[4]
1140 Erc% = DcCommand(endit%, PTR (Buf%[1]), zero%)
1150 IF Erc% <> ErcOk% GOTO 1140
1155 '
1160 END
1170 '
1499 '**************************************
1500 '**  subroutine for Checking the Erc  **
1501 '**************************************
1550 PRINT messag$, " ", Erc%
1600 END
```

# PASCAL ECHO PROGRAM
# USING MULTIPLE TASK INTERFACE

NOTE: The Multiple-Task Interface is highly sensitive to timing when more than one address is used. This program may or may not work, when 2 or 3 addresses are used, depending upon the exact sequence of polls and selects issued by the host.

```
{$DEBUG-}
{$ENTRY-}

PROGRAM MpEcho (INPUT, OUTPUT);

TYPE
    String2  = STRING(2);
    pointr   = ADS OF BYTE;

CONST

 (* Command codes for multiple task interface *)
    Configc  = 16#0000;
    Xfer_Rec_Bufc = 16#0001;
    Xfer_Xmt_Bufc = 16#0002;
    Offlinec  = 16#0003;
    Onlinec  = 16#0004;
    Idlec  = 16#0005;
    Fastsetc  = 16#0006;
    Receivec  = 16#0007;
    Transmitc  = 16#0008;
    Endsessionc  = 16#0009;
    Abortc  = 16#000A;
    Fastresetc  = 16#000B;

 (* Erc return codes for multiple task interface *)
    Cmd_Accepted  = 16#0000;
    Task_oflow  = 16#8001;
    Cmd_Pending  = 16#8002;
    Rpt_Pending  = 16#8003;
    Invalid_Addr  = 16#8004;
    Cmd_Denied  = 16#8005;
    Buffer_oflow  = 16#8006;
    Report_error  = 16#8007;
```

```
 (* Report codes *)
    No_report   = 16#0000;
    Receiving   = 16#0001;
    Rec_Grp_Sel    = 16#0002;
    Receive_Done   = 16#0003;
    Rdy_Xmt_Xfer   = 16#0004;
    Transmit_Done  = 16#0005;
    Select_Denied  = 16#0006;
    Receive_err    = 16#0007;
    Dup_seq_num    = 16#0008;
    Seq_num_err    = 16#0009;
    Transmit_err   = 16#000A;
    Internal_err   = 16#00FF;

VAR [PUBLIC]
    Erc             : WORD;
    numAdrs         : INTEGER;
    etx             : CHAR;
    counter1        : INTEGER;
    counter2        : INTEGER;
    i               : INTEGER;
    Report          : ARRAY [0..2] OF WORD;
    DevAdr          : ARRAY [0..2] OF String2;
    Buff            : ARRAY [0..6143] OF CHAR;
    sBuf            : ARRAY [0..2] OF INTEGER;
    sBufMax         : INTEGER;
    LogMsg          : STRING(20);
    sLogMsg         : INTEGER;

VALUE
    etx             := CHR (03);
    sBufMax         := 2048;
    LogMsg          := '?ASSIGN ECHOPROG    ';
    sLogMsg         := 16;

FUNCTION MpReport (Addr       : String2;
                   pReportRet : pointr) : WORD; EXTERN;

FUNCTION MpReportWait (Addr       : String2;
                       pReportRet : pointr) : WORD; EXTERN;

FUNCTION MpCommand (comm    : WORD;
                    Addr    : String2;
                    pbuffer : pointr;
                    sbuffer : INTEGER) : WORD; EXTERN;

PROCEDURE Exit; EXTERN;

PROCEDURE Error;
BEGIN
  WRITELN ('Command Denied', Erc);
  Exit;
END;
```

```
FUNCTION Get_Buffer (int : INTEGER) : BOOLEAN;
BEGIN
    Get_Buffer := FALSE;
    Erc := MpCommand (Xfer_Rec_Bufc, DevAdr [int],
                      ADS buff [int * 2048], sBufMax);
    IF (Erc <> Cmd_Accepted) THEN RETURN;

    Get_Buffer := TRUE;
    sBuf [int] := 0;
    WHILE (buff [sBuf [int] + int * 2048] <> etx)
        AND (sBuf [int] < sBufMax)
            DO
               sBuf [int] := sBuf [int] + 1;

END; (* PROCEDURE Get_Buffer *)


PROCEDURE Go_Transmit_Ready (int : INTEGER);
BEGIN
    Erc := MpCommand (Transmitc, DevAdr [int],
                      ADS buff [0], 0);
    IF (Erc <> Cmd_Accepted) THEN Error;

END;  (* PROCEDURE Go_Transmit_Ready *)


PROCEDURE Put_Buffer (int : INTEGER);
BEGIN
    Erc := MpCommand (Xfer_Xmt_Bufc, DevAdr [int],
                      ADS buff [int * 2048], sBuf [int]);
    IF (Erc <> Cmd_Accepted) THEN Error;

END;  (* PROCEDURE Put_Buffer *)


PROCEDURE Go_Receive_Ready (int : INTEGER);
BEGIN
    Erc := MpCommand (Receivec, DevAdr [int],
                      ADS buff[0], 0);
    IF (Erc <> Cmd_Accepted) THEN Error;

END;  (* PROCEDURE Go_Receive_Ready *)
```

```
PROCEDURE Transmit_Signon (int : INTEGER);
BEGIN
    FOR counter2 := Ø TO sLogMsg DO
        Buff [int * 2Ø48 + counter2] := LogMsg [counter2];

    Go_Transmit_Ready (int);
    REPEAT
        erc := MpReport (DevAdr [int], ADS Report [int]);
    UNTIL (Report [int] = Rdy_Xmt_Xfer);

    Put_Buffer (int);
    REPEAT
        erc := MpReport (DevAdr [int], ADS Report [int]);
    UNTIL (Report [int] = Transmit_Done);

    Go_Receive_Ready (int);

END;  (* PROCEDURE Transmit_Signon *)

PROCEDURE Configure (int : INTEGER);
BEGIN
    REPEAT

      WRITE  ('Enter Station Address: ');
      READLN (DevAdr [int] );
      Erc := MpCommand (Configc, DevAdr [int],
                        ADS buff [Ø], Ø);

    UNTIL (Erc = Cmd_Accepted);

    Erc := MpCommand (Onlinec, DevAdr [int],
                      ADS buff [Ø], Ø);
    IF (Erc <> Cmd_Accepted) THEN Error;

END;  (* PROCEDURE Configure *)

BEGIN
   WRITE   ('BMULTI echo program ');
   WRITELN ('using multiple address interface ');
   WRITELN (' ');
   WRITELN ('How many addresses do you desire? ');
   READLN  (numAdrs);
   IF (numAdrs > 3) THEN numAdrs := 3;
   IF (numAdrs < 1) THEN numAdrs := 1;
   numAdrs := numAdrs - 1;

   FOR i := Ø TO numAdrs DO Configure (i);

   FOR i := Ø TO numAdrs DO Transmit_Signon (i);
```

```
    counter1 := 1;
    WHILE counter1 < 50 DO
      BEGIN

        FOR i := 0 TO numAdrs DO
          BEGIN

            REPEAT
              Erc := MpReport (DevAdr [i], ADS Report [i]);
            UNTIL (Erc = Cmd_Accepted);

            CASE Report [i] OF

              Receive_Done, Dup_seq_num, Seq_num_err:

                IF (Get_Buffer (i) = TRUE)
                  THEN
                    BEGIN
                      counter1 := counter1 + 1;
                      Go_Transmit_Ready (i);
                    END;

              Rdy_Xmt_Xfer:  Put_Buffer (i);

              Transmit_Done:  Go_Receive_Ready (i);

              No_report:      ;  (* Do nothing *)
              Receiving:      ;
              Rec_Grp_Sel:    ;
              Select_Denied:  ;
              Receive_err:    ;
              Transmit_err:   ;

            END;  (* CASE Report OF *)

          END;  (* FOR i := 0 TO numAdrs DO *)

      END;  (* WHILE counter1 < 50 *)

    WRITELN ('End BMULTI Echo Program ');

    FOR i := 0 TO numAdrs DO
      BEGIN
        REPEAT
          Erc := MpCommand (Idlec, DevAdr [i],
                            ADS buff [0], 0);
          Erc := MpCommand (Endsessionc, DevAdr [i],
                            ADS buff [0], 0);
        UNTIL  (Erc = Cmd_Accepted);
      END;

    Exit;
  END.
```

# PASCAL TERMINAL PROGRAM
# USING ENHANCED LOW-LEVEL INTERFACE

```
{$DEBUG-}
{$ENTRY-}

PROGRAM MiniTerm;

TYPE
    String2  = STRING(2);
    pbType   = ADS OF BYTE;
    pwType   = ADS OF WORD;
    ppType   = ADS OF pbType;
    psType   = ADS OF String2;

CONST

 (* Command codes for new low-level interface *)
    Xfer_Rec_Bufc = 16#0001;
    Xfer_Xmt_Bufc = 16#0002;
    Offlinec      = 16#0003;
    Onlinec       = 16#0004;
    Idlec         = 16#0005;
    Fastsetc      = 16#0006;
    Receivec      = 16#0007;
    Transmitc     = 16#0008;
    Endsessionc   = 16#0009;
    Abortc        = 16#000A;
    Fastresetc    = 16#000B;
    Xmt_Big_Bufc  = 16#000C;

 (* Report codes *)
    No_report     = 16#0000;
    Receiving     = 16#0001;
    Rec_Grp_Sel   = 16#0002;
    Receive_Done  = 16#0003;
    Rdy_Xmt_Xfer  = 16#0004;
    Transmit_Done = 16#0005;
    Select_Denied = 16#0006;
    Receive_err   = 16#0007;
    Dup_seq_num   = 16#0008;
    Seq_num_err   = 16#0009;
    Transmit_err  = 16#000A;
    Internal_err  = 16#00FF;

 (* Erc return codes for new low-level interface *)
    ErcOk               = 16#0000;
    ErcInvalidCmd       = 16#8000;
    ErcTaskOverflow     = 16#8001;
    ErcCmdPending       = 16#8002;
    ErcReportPending    = 16#8003;
    ErcInvalidAddress   = 16#8004;
    ErcCmdDenied        = 16#8005;
    ErcBufferOverflow   = 16#8006;
```

```
    ErcInvalidReportRq  = 16#8007;
    ErcReadInProgress   = 16#8008;
    ErcWriteInProgress  = 16#8009;
    ErcBufferInUse      = 16#800A;
    ErcInvalidBufLength = 16#800B;
    ErcOfflineDenied    = 16#800C;
    ErcOnlineDenied     = 16#800D;
    ErcIdleDenied       = 16#800E;
    ErcFastRdyDenied    = 16#800F;
    ErcXmtRdyDenied     = 16#8010;
    ErcRcvRdyDenied     = 16#8011;
    ErcXfrXmtDenied     = 16#8012;
    ErcXfrRcvDenied     = 16#8013;
    ErcEndSessDenied    = 16#8014;
    ErcNotOnline        = 16#8015;
    ErcStationOverflow  = 16#8016;
    ErcAddrIsGrpAddr    = 16#8017;
    ErcIncompleteMsg    = 16#8018;
    ErcInternalError    = 16#8019;
    ErcDupVirtualAdr    = 16#801A;
    ErcReconfiguration  = 16#801B;
    ErcEntryError       = 16#801C;
    ErcStationActive    = 16#801D;

 (* Miscellaneous*)
    banner          = 'B20 Mini-Term';
    sBanner         = 19;

VAR [PUBLIC]
    Erc             : WORD;
    Report          : WORD;
    th              : WORD;
    DevAdr          : String2;
    cMsg            : INTEGER;
    Msg             : ARRAY[0..79] OF BYTE;
    sBufMax         : INTEGER;
    sBuf            : INTEGER;
    Buff            : ARRAY[0..2047] OF CHAR;
    dummyPtr        : pbType;
    pVidSeg         : pbType;
    sMap            : WORD;
    nLines          : INTEGER;
    Key             : BYTE;
    current_col     : INTEGER;
    vid_col         : INTEGER;
    SdRet           : RECORD
        pSubParam : psType;
        sSubParam : WORD;
                    END;
```

```
    vHdw          : RECORD
        level     : BYTE;
        nLinesMax : SINT;
        nColsNar  : BYTE;
        nColsWide : BYTE;
                    END;

VALUE
    sBufMax       := 4096;
    sMap          := 16#0B6C;

!#############################################################
!#          System Common Procedures
!#############################################################
PROCEDURE Exit; EXTERN;

PROCEDURE ErrorExit (ercTerm : WORD); EXTERN;

FUNCTION  PosFrameCursor (iFrame : INTEGER;
                          iCol   : INTEGER;
                          iLine  : INTEGER) : WORD; EXTERN;

FUNCTION  PutFrameChars (iFrame : INTEGER;
                         iCol   : INTEGER;
                         iLine  : INTEGER;
                         pbText : pbType;
                         cbText : INTEGER) : WORD; EXTERN;

FUNCTION  ResetFrame ( iFrame : INTEGER) : WORD; EXTERN;

FUNCTION  ScrollFrame ( iFrame     : INTEGER;
                        iLineStart : INTEGER;
                        iLineMax   : INTEGER;
                        cLines     : INTEGER;
                        fUp        : BOOLEAN) : WORD; EXTERN;

!#############################################################
!#          Object Module Procedures
!#############################################################
FUNCTION BmOpen (Addr   : String2;
                 pTaskH : pwType;
                 fSys   : BOOLEAN) : WORD; EXTERN;

FUNCTION BmReportWait (TaskH      : WORD;
                       pReportRe' : pwType) : WORD; EXTERN;

FUNCTION BmReport (TaskH      : WORD;
                   pReportRet : pwType) : WORD; EXTERN;

FUNCTION BmCommand (TaskH   : WORD;
                    Comm    : WORD;
                    pbuffer : pbType;
                    sbuffer : INTEGER) : WORD; EXTERN;
```

```
FUNCTION  RgParam (iParam    : WORD;
                   iSubParam : WORD;
                   pSdRet    : pbType) : WORD; EXTERN;

!#########################################################
!#          Procedural requests
!#########################################################
FUNCTION  InitCharMap (pMap : pbType;
                       sMap : WORD) : WORD; EXTERN;

FUNCTION  InitVidFrame (iFrame      : INTEGER;
                        iColStart   : INTEGER;
                        iLineStart  : INTEGER;
                        nCols       : INTEGER;
                        nLines      : INTEGER;
                        borderDesc  : BYTE;
                        bBorderChar : CHAR;
                        bBorderAttr : BYTE;
                        fDblHigh    : BOOLEAN;
                        fDblWide    : BOOLEAN) : WORD; EXTERN;

FUNCTION  QueryVidHdw (pBuf : pbType;
                       sBuf : WORD) : WORD; EXTERN;

FUNCTION  ReadKbdDirect (mode     : WORD;
                         pCharRet : pbType) : WORD; EXTERN;

FUNCTION  ResetVideo (nCols    : INTEGER;
                      nLines   : INTEGER;
                      fAttr    : BOOLEAN;
                      bSpace   : CHAR;
                      psMapRet : pbType) : WORD; EXTERN;

FUNCTION  SetScreenVidAttr (iAttr : WORD;
                            fOn   : BOOLEAN) : WORD; EXTERN;

!#########################################################
  PROCEDURE Check_Erc (Irk : WORD)  [PUBLIC];
!#########################################################
BEGIN
  IF (Irk <> 0) THEN ErrorExit (Irk);
END;
```

```
!#############################################################
  PROCEDURE Screen_setup  [PUBLIC];
!#############################################################
!#   Initialized the video.
!#############################################################
VAR
  BannerStart : INTEGER;
BEGIN
  Check_Erc (QueryVidHdw (ADS vHdw, 4));
  nLines := vHdw.nLinesMax;
  Check_Erc (ResetVideo (80, nLines, FALSE,
                         ' ', ADS sMap));
  Check_Erc (InitVidFrame (0, 0, 0, 80, 1,
                           4, ' ', 0, FALSE, FALSE));
  Check_Erc (InitVidFrame (1, 0, 2, 80, nLines - 3,
                           0, ' ', 0, FALSE, FALSE));
  Check_Erc (InitVidFrame (2, 0, 2, 80, nLines - 2,
                           0, ' ', 0, FALSE, FALSE));
  pVidSeg.s := 0;
  pVidSeg.r := 0;
  Check_Erc (InitCharMap (pVidSeg, sMap));
  Check_Erc (SetScreenVidAttr (1, TRUE));
  Check_Erc (PosFrameCursor (2, 0, nLines - 3));
  current_col := 0;
  BannerStart := (80 - sBanner ) DIV 2;
  Check_Erc (PutFrameChars (0, BannerStart, 0,
                            ADS banner, sBanner));
END; (* PROCEDURE Screen_setup *)

!#############################################################
  PROCEDURE Process_Dcom_input  [PUBLIC];
!#############################################################
VAR
  i     : INTEGER;
BEGIN
  Check_Erc (ScrollFrame (1, 0, 255, 1, TRUE));
  Check_Erc (PosFrameCursor (1, 255, 255));
  vid_col := 0;
  FOR i := 0 TO sBuf DO
    BEGIN
      Check_Erc (PutFrameChars (1, vid_col, nLines - 4,
                                ADS buff [i], 1));
      vid_col := vid_col + 1;
      IF (vid_col > 79)
        THEN
          BEGIN
            Check_Erc (ScrollFrame (1, 0, 255, 1, TRUE));
            Check_Erc (PosFrameCursor (1, 255, 255));
            vid_col := 0;
          END;
    END;
END; (* PROCEDURE Process_Dcom_input *)
```

```
!#########################################################
  PROCEDURE Process_Kbd_input  [PUBLIC];
!#########################################################
BEGIN
  IF (Key = 8)    !8=BACKSPACE
    THEN
      BEGIN
        IF (current_col = 79)
          THEN Check_Erc (PutFrameChars (2, 79,
                          nLines - 3, ADS ' ', 1));
        IF (current_col > 0)
          THEN current_col := current_col - 1;
        Check_Erc (PutFrameChars (2, current_col, nLines - 3,
                                  ADS ' ', 1));
        Check_Erc (PosFrameCursor (2, current_col, nLines - 3));
      END
    ELSE
      BEGIN
        Msg [current_col] := Key;
        Check_Erc (PutFrameChars (2, current_col, nLines - 3,
                                  ADS Key, 1));
        IF (current_col < 79)
          THEN current_col := current_col + 1;
        Check_Erc (PosFrameCursor (2, current_col, nLines - 3));
      END;
END; (* PROCEDURE Process_Kbd_input *)

!#########################################################
  PROCEDURE Active_state  [PUBLIC];
!#########################################################
!#   This is the main loop of the program.  It
!#   alternately checks the Bmulti report queue
!#   and the keyboard queue for activity.
!#########################################################
VAR
  loop1       : BOOLEAN;
  loop2       : BOOLEAN;

BEGIN
  WHILE TRUE DO
  BEGIN
    loop1 := TRUE;
    WHILE loop1 DO
    BEGIN
      Erc := BmReport (th, ADS Report);
      IF (Erc = ErcOk)
        THEN
          CASE Report OF

            No_report:  loop1 := FALSE;

            Transmit_Done:
                Erc := BmCommand (th, Receivec, dummyPtr, 0);
```

```
            Rdy_Xmt_Xfer:
              Erc := BmCommand (th, Xfer_Xmt_Bufc,
                                ADS Msg, cMsg);

            Receive_Done, Dup_seq_num, Seq_num_err:
              BEGIN
                Erc := BmCommand (th, Xfer_Rec_Bufc,
                                  ADS sBuf, sBufMax);
                IF (Erc = ErcOk)
                  THEN
                    Process_Dcom_input;
                Erc := BmCommand (th, Receivec, dummyPtr, 0);
              END;
          END;     (* CASE Report OF *)
    END;    (* WHILE loop1 DO *)

    loop2 := TRUE;
    WHILE loop2 DO
    BEGIN
      Erc := ReadKbdDirect (1, ADS Key);
      IF (Erc = 602)
        THEN loop2 := FALSE
        ELSE
          BEGIN
            Erc := BmCommand (th, Idlec, dummyPtr, 0);
            CASE Key OF

            4:  RETURN;  !FINISH key

            10, 27:      !RETURN, NEXT, and GO keys
               BEGIN
                 cMsg := current_col;
                 Erc := BmCommand (th, Idlec, dummyPtr, 0);
                 Erc := BmCommand (th, Transmitc, dummyPtr, 0);
                 IF (Erc = ErcOk)
                   THEN
                     BEGIN
                       Check_Erc (ScrollFrame (2, 0, 255, 1, TRUE));
                       Check_Erc (PosFrameCursor (2, 0, nLines - 3));
                       current_col := 0;
                     END
               END

            OTHERWISE   Process_Kbd_input;
           END;

          END;  (* CASE Key OF *)
      END;    (* loop2  *)
    END;    (* WHILE TRUE DO *)

END;    (* PROCEDURE Active_state *)
!##############################################################
!#          MAIN PROGRAM
!##############################################################
```

```
BEGIN

!#############################################################
!#  Retrieve Device Address as either parameter 1 or 2,
!#  depending on whether Run File command is used, or
!#  the program's own command.
!#############################################################
  Check_Erc (RgParam (1, 0, ADS SdRet));
  IF (SdRet.sSubParam <> 2)
    THEN  Check_Erc (RgParam (2, 0, ADS SdRet));
  DevAdr := SdRet.pSubParam?;

!#############################################################
!#  Initialize the video.
!#############################################################
  Screen_setup;

!#############################################################
!#  Log onto Bmulti with the Device Address.
!#############################################################
  Erc := BmOpen (DevAdr, ADS th, FALSE);
  IF (Erc <> ErcOk)
    THEN ErrorExit (Erc);
  Erc := BmCommand (th, Onlinec, dummyPtr, 0);

!#############################################################
!#  Enter an infinite loop.
!#############################################################
  Active_state;     !Does not return until FINISH is hit.

!#############################################################
!#  At termination, deallocate resources.
!#############################################################
  REPEAT
    Erc := BmCommand (th, Idlec, dummyPtr, 0);
  UNTIL (Erc = ErcOk);
  Erc := BmCommand (th, Endsessionc, dummyPtr, 0);
  Exit;
END.       (* PROGRAM MiniTermBm *)
```

# COBOL ECHO PROGRAM
# USING HIGH-LEVEL INTERFACE

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  Jim.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  B20.
 OBJECT-COMPUTER.  B20.
 DATA DIVISION.
 FILE SECTION.
 WORKING-STORAGE SECTION.
 77  counter                PIC 9(04) COMP.
 01  miscellaneous.
     03  device-address     PIC XX.
     03  ercdisplay         PIC 9(04).
     03  max-buffer-size    PIC 9(04) COMP VALUE 4096.
     03  online             PIC X VALUE X"00".
     03  fss                PIC X.
     03  pri                PIC 9(04) COMP.
     03  TskH               PIC X.
     03  buffer-size        PIC 9(04) COMP.
     03  buffer-size-a2     REDEFINES buffer-size PIC X OCCURS 2.
*
 01  buffer-whole.
     03  buffer-size-a      PIC X  OCCURS 2.
     03  buffer             PIC X(4096).
     03  buffer-array       REDEFINES buffer.
         05  byte           PIC X  OCCURS 4096.
*
 01  Error-code             PIC 9(04) COMP.
     88  Address-Is-Good    VALUE 0.
     88  No-Error           VALUE 0.
*
 01  asBlk.
     03  RcvStatus          PIC X.
         88  Idle           VALUE X"00".
         88  ReadBusy       VALUE X"01".
         88  ReadErr        VALUE X"02".
         88  ReadDone       VALUE X"10".
   88  SeqErr         VALUE X"11".
   88  DupSeq         VALUE X"12".
         88  TruncMsg       VALUE X"14".
     03  RcvErc             PIC 9(04) COMP.
     03  fSelDen            PIC X.
     03  XmtStatus          PIC X.
         88  Local          VALUE X"00".
         88  WriteBusy      VALUE X"01".
         88  WriteErr       VALUE X"02".
         88  WriteDone      VALUE X"10".
     03  XmtErc             PIC 9(04) COMP.
     03  Opt                PIC X.
     03  fMess              PIC X.
     03  pfMess             PIC X(04).
```

```
*
 PROCEDURE DIVISION.
 MAIN-LINE.
      PERFORM start-up.
      PERFORM driver THRU driver-x
          VARYING counter FROM 1 BY 1
              UNTIL counter IS EQUAL TO 50.
      PERFORM finish-up.
*
 START-UP.
      DISPLAY "BMULTI echo program" UPON CONSOLE.
      PERFORM Get-Address THRU Get-Address-X
          UNTIL Address-Is-Good.
      DISPLAY "Begin BMULTI" UPON CONSOLE.
      CALL "&SETOPTIONBMULTI" USING error-code, TskH, online.
      PERFORM error-check.
*
 DRIVER.
      MOVE SPACES TO buffer.
      CALL "&READBMULTI" USING error-code, TskH, buffer-whole,
                               max-buffer-size.
      PERFORM Error-Check.
      PERFORM Check-Read-Complete.
      MOVE buffer-size-a (1) TO buffer-size-a2 (2).
      MOVE buffer-size-a (2) TO buffer-size-a2 (1).
      MOVE buffer-size TO ercdisplay.
      DISPLAY "Message size is: ",ercdisplay UPON CONSOLE.
      DISPLAY "Counter is  ",counter UPON CONSOLE.
      CALL "&WRITEBMULTI" USING error-code, TskH, buffer,
                                buffer-size.
      PERFORM Error-Check.
      PERFORM Check-Write-Complete.
*
 DRIVER-X.
      EXIT.
*
 FINISH-UP.
      CALL "&CLOSEBMULTI" USING error-code, TskH.
      IF NOT No-Error
          GO TO finish-up.
      STOP RUN.
*
 GET-ADDRESS.
      DISPLAY "Enter Station Address: " UPON CONSOLE.
      ACCEPT device-address.
      CALL "&OPENBMULTI" USING error-code, device-address,
                               fss, pri, TskH, asBlk.
      IF NOT No-Error
          PERFORM print-error.
*
 GET-ADDRESS-X.
      EXIT.
```

```
*
 CHECK-READ-COMPLETE.
      IF NOT ReadDone
          GO TO CHECK-READ-COMPLETE.
*
 CHECK-WRITE-COMPLETE.
      IF NOT WriteDone
          GO TO CHECK-WRITE-COMPLETE.
*
 ERROR-CHECK.
      IF NOT No-Error
          PERFORM print-error.
*
 PRINT-ERROR.
      MOVE error-code TO ercdisplay.
      DISPLAY "BMULTI error ", ercdisplay UPON CONSOLE.
      STOP RUN.
*
 END-OF-JOB.
```

# APPENDIX B
# USASCII CODE CHARTS

| b7 b6 b5 / BITS | b4 | b3 | b2 | b1 | ROW \ COLUMN | 0 (0 0 0) | 1 (0 0 1) | 2 (0 1 0) | 3 (0 1 1) | 4 (1 0 0) | 5 (1 0 1) | 6 (1 1 0) | 7 (1 1 1) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | NUL | DLE | | 0 | | | | POL |
| | 0 | 0 | 0 | 1 | 1 | SOH | DC1 | | 1 | | | | SEL |
| | 0 | 0 | 1 | 0 | 2 | STX | DC2 | | | | | | |
| | 0 | 0 | 1 | 1 | 3 | ETX | DC3 | | | | | | FSL |
| | 0 | 1 | 0 | 0 | 4 | EOT | DC4 | | | | | | BSL |
| | 0 | 1 | 0 | 1 | 5 | ENQ | NAK | | | | | | |
| | 0 | 1 | 1 | 0 | 6 | ACK | SYN | | | | | | |
| | 0 | 1 | 1 | 1 | 7 | BEL* | ETB | | | | | | |
| | 1 | 0 | 0 | 0 | 8 | BS | CAN | | | | | | |
| | 1 | 0 | 0 | 1 | 9 | HT | EM | | | | | | |
| | 1 | 0 | 1 | 0 | 10 | LF | SUB | | | | | | |
| | 1 | 0 | 1 | 1 | 11 | VT | ESC | | | | [ | | |
| | 1 | 1 | 0 | 0 | 12 | FF | FS | | < | | | | |
| | 1 | 1 | 0 | 1 | 13 | CR | GS | | | | ] | | |
| | 1 | 1 | 1 | 0 | 14 | SO | RS | | | | | | |
| | 1 | 1 | 1 | 1 | 15 | SI | US | | | | | | DEL |

* CON (ALTERNATE CODE FOR CONTENTION)

**Figure B-1. Code Chart**

**Showing Universal Control Codes Plus Special Allocation of Codes To Implement the Burroughs Multipoint Protocol**

| BITS b7 b6 b5 | | | | | 0 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1 0 0 | 1 0 1 | 1 1 0 | 1 1 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b4 | b3 | b2 | b1 | ROW \ COLUMN | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 0 | 0 | 0 | 0 | NUL | DLE | SP | 0 | @ | P | ` | p |
| 0 | 0 | 0 | 1 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0 | 0 | 1 | 0 | 2 | STX | DC2 | " | 2 | B | R | b | r |
| 0 | 0 | 1 | 1 | 3 | ETX | DC3 | # | 3 | C | S | c | s |
| 0 | 1 | 0 | 0 | 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0 | 1 | 0 | 1 | 5 | ENQ | NAK | % | 5 | E | U | e | u |
| 0 | 1 | 1 | 0 | 6 | ACK | SYN | & | 6 | F | V | f | v |
| 0 | 1 | 1 | 1 | 7 | BEI | ETB | ' | 7 | G | W | g | w |
| 1 | 0 | 0 | 0 | 8 | BS | CAN | ( | 8 | H | X | h | x |
| 1 | 0 | 0 | 1 | 9 | HT | EM | ) | 9 | I | Y | i | y |
| 1 | 0 | 1 | 0 | 10 | LF | SUB | * | : | J | Z | j | z |
| 1 | 0 | 1 | 1 | 11 | VT | ESC | + | ; | K | [ | k | { |
| 1 | 1 | 0 | 0 | 12 | FF | FS | , | < | L | \ | l | \| |
| 1 | 1 | 0 | 1 | 13 | CR | GS | - | = | M | ] | m | } |
| 1 | 1 | 1 | 0 | 14 | SO | RS | . | > | N | ^ | n | ~ |
| 1 | 1 | 1 | 1 | 15 | SI | US | / | ? | O | _ | o | DEL |

**Figure B-2. USA Standard Code for Information Interchange**

# APPENDIX C
# HARDWARE CONSIDERATIONS

In order to use BMULTI, the B 20 or XE 520 must be connected to a Burroughs mainframe. Typically, the connection is through a direct line, a leased line, or the switched telephone network. Additional hardware is necessary to connect a B 20 or XE 520 to any of these lines.

The B 20 and XE 520 communication hardware uses the RS-232 interface. This means that the B 20 or XE 520 must be cabled to a modem or a Burroughs TDI/Concatenation Adaptor (B 20 DCA).

A Burroughs TDI/Concatenation Adaptor is required in order to connect a B 20 or an XE 520 to a Burroughs Two-wire Direct Interface (TDI) network. The DCA provides TDI according to the setting of the TDI/concatenation switch on the front panel.

If a B 20 is to be connected to a modem line with other Burroughs terminals, a DCA is required in order to place the B 20 anywhere other than last in the concatenation string. (This type of connection is not supported on the XE 520.) While set for concatenation, the adaptor turns ON (applies a positive voltage to) pin 16 if downstream Request-to-Send in ON. If the B 20 turns OFF (applies a negative voltage to) pin 14, the adaptor blocks Request-to-Send from downstream terminals and Clear-to-Send from the modem. A second switch on the front of the adaptor sets the Rate Select and Select Standby signals to specified levels.

Modems for use with BMULTI may be either synchronous or asynchronous, 2-wire or 4-wire. Modems may be purchased from Burroughs or another vendor.

**NOTE**

If you are installing a B 20 or XE 520 which will be connected to an existing line, you must select the B 20 or XE 520 modem to match the host modem characteristics.

In many systems, modem options are dictated by the conditions of the line to which the B 20 or XE 520 is connected. Where other considerations permit, Burroughs recommends the following modem option settings:

* transmitter internally timed
* 4-wire operation
* switched carrier
* without new synchronization

For synchronous operation with an internally timed modem, the B 22 system switches must be set for external clock, meaning clocking that is external to the workstation but internal to the modem. (There are no internal switches on the B 21, B 25, or XE 520.)

Assuming the modem is connected to communications channel B, the required switch settings on the I/O-memory board are shown in table C-1.

**Table C-1. Switch Settings for Channel B on I/O-Memory Board (Switch Box 1)**

| Switch | Sync Setting | Async/TDI Setting |
|---|---|---|
| 5 | ON | OFF |
| 6 | ON | OFF |
| 7 | OFF | ON |
| 8 | OFF | ON |

A double-male RS-232 extension cable must be used to connect the B 20 or XE 520 to the modem. It should be a straight-through terminal-to-modem cable rather than the crossover (null modem) type.

RS-232C signals used in operation are shown in table C-2 below. Those used only in synchronous operation are so marked.

Table C-2. RS-232C Signals in Operation

| Pin number | Signal Name |
|---|---|
| 1 | Protective Ground |
| 7 | Signal Ground |
| 2 | Transmit Data |
| 3 | Receive Data |
| 4 | Request to Send (RTS) |
| 5 | Clear to Send (CTS) |
| 6 | Data Set Ready |
| 8 | Data Carrier Detect (Not used on B 20) |
| 14 | Block Downstream CTS (Not used on XE 520) |
| 15 | Transmit Clock (Sync only) |
| 16 | Sense Downstream RTS (Not used on XE 520) |
| 17 | Receive Clock (Sync only) |
| 20 | Data Terminal Ready |

When using B 20s in a concatenation environment (which requires the B 20 DCA), upstream B 20s must be turned on in order for downstream terminals or B 20s to communicate with the host.

The RS-232 cable shipped with the B 20 DCA box should not be used for any connections other than from the DCA to the B 20 or XE 520. Certain pins exist in the cable that are used for other purposes by other Burroughs terminals, and any use of this cable with other terminals might result in unpredictable results.

About 15 seconds elapse between powering on a B 22 and the beginning of Operating System execution. During this period, the B 22 has RTS (RS-232 pin 4) turned ON. This may termporarily inhibit communication in switched-line environments.

# APPENDIX D
# LANGUAGE CONFIGURATION

FORTRAN, COBOL, AND BASIC cannot be used with BMULTI without regenerating the language interpreters and/or libraries. This is done by using the Editor to add certain lines to the ".asm" file associated with the language desired (either 'CobolGen.asm', 'BasGen.asm', or 'ForGen.asm'), Assembling it, and either re-linking the interpreter or, in the case of FORTRAN and compiled BASIC, linking the resultant object module with the object module which resulted from the compile.

The following lines must be added to the appropriate ".asm" file at the locations indicated by comments in each file, below the comment ADD NEW ENTRIES HERE:

BASIC (BasGen.asm)

```
%TableEntry(1,14,OPENBMULTI)
%TableEntry(1,8,READBMULTI)
%TableEntry(1,8,WRITEBMULTI)
%TableEntry(1,4,SETOPTIONBMULTI)
%TableEntry(1,2,RESETBMULTI)
%TableEntry(1,2,CLOSEBMULTI)
```

COBOL (CobolGen.asm)

```
%TableEntry(0,w,OPENBMULTI,5,w,b,w,r,r)
%TableEntry(0,w,READBMULTI,3,b,r,w)
%TableEntry(0,w,WRITEBMULTI,3,b,r,w)
%TableEntry(0,w,SETOPTIONBMULTI,2,b,b)
%TableEntry(0,w,RESETBMULTI,1,b)
%TableEntry(0,w,CLOSEBMULTI,1,b)
```

FORTRAN (ForGen.asm)

```
%TableEntry(OPENBMULTI,BMULTO,5,w,b,w,r,r)
%TableEntry(READBMULTI,BMULTR,3,b,r,w)
%TableEntry(WRITEBMULTI,BMULTW,3,b,r,w)
%TableEntry(SETOPTIONBMULTI,BMULTS,2,b,b)
%TableEntry(RESETBMULTI,BMULTT,1,b)
%TableEntry(CLOSEBMULTI,BMULTC,1,b)
```

For directions on assembling and linking these files, see the B 20 Systems Reference Manual for the appropriate language.

# APPENDIX E

# BTOS REQUEST CODES FOR BMULTI

BMulti uses request codes -1 through -7. All system software releases numbered 4.0 and higher support these requests. System releases numbered 2.3 through 4.0 support only request codes -1 and -2. System software releases 1.3 and lower do not support BMulti.

Near the end of the file, in the user request code table:

```
%UsrRequest(-1,DCCommand,exchNotInstalled,0000h,3,1,1,
            %( %illegal))
%UsrRequest(-2,DCReport,exchNotInstalled,0000h,2,0,1,
            %( %illegal))
%UsrRequest(-3,BMCommand,exchNotInstalled,000h4,1,1,
            %( %illegal))
%UsrRequest(-4,BMReport,exchNotInstalled,000h,2,0,1,
            %( %illegal))
%UsrRequest(-5,BMQuery,exchNotInstalled,000h,0,0,1,
            %( %illegal))
%UsrRequest(-6,BMClear,exchNotInstalled,000h,2,0,0,
            %( %none )
            %( %norouting ))
%UsrRequest(-7,BMPurge,exchNotInstalled,000h,2,0,0,
            %( %none )
            %( %norouting ))
```

Note: Burroughs reserves user request codes -8 through -10 for future development.

The file Request.asm must then be assembled in the manner described in the System Programmer's Guide, Volume 1, or the customizer technical notes for creating a new operating system.

# APPENDIX F
# STATUS CODES GENERATED BY ENHANCED LOW-LEVEL INTERFACE

| ERROR CODE (HEX) | ERROR CODE (DECIMAL) | EXPLANATION |
|---|---|---|
| 8000 | 32768 | Invalid command was issued to Bmulti. |
| 8001 | 32769 | Task Overflow. The multi tasking interface cannot handle more than three device addresses. |
| 8002 | 32770 | Command Pending. Only one Bmulti command can be outstanding at any time per address. |
| 8003 | 32771 | Report Pending. Only one Bmulti report request can be outstanding at any time for an address. |
| 8004 | 32772 | Invalid Bmulti address. |
| 8005 | 32773 | Command Denied (Mp Interface). |
| 8006 | 32774 | Buffer Overflow. |
| 8007 | 32775 | Invalid Report Request. |
| 8008 | 32776 | Read in Progress. |
| 8009 | 32777 | Write in Progress. |
| 800A | 32778 | A request for a large buffer transfer cannot be honored at this time for want of buffer space. Try again later. |
| 800B | 32779 | Invalid buffer length for this command. |
| 800C | 32780 | Offline Command Denied. |
| 800D | 32781 | Online Command Denied. |
| 800E | 32782 | Idle Command Denied. |
| 800F | 32783 | Fast Rdy Command Denied. |
| 8010 | 32784 | Transmit Rdy Command Denied. |
| 8011 | 32785 | Receive Rdy Command Denied. |
| 8012 | 32786 | Transfer Xmt Buffer Command Denied. |
| 8013 | 32787 | Transfer Rcv Buffer Command Denied. |
| 8014 | 32788 | End Session Command Denied. |
| 8015 | 32789 | Station Not Online. |
| 8016 | 32790 | Station Overflow. |
| 8017 | 32791 | Device Address Clashes with Group Address. |
| 8018 | 32792 | The buffer received by the application is only part of the full message. |
| 8019 | 32793 | Internal error (Please report to the system administrator). |
| 801A | 32794 | Duplicate Virtual Address. |
| 801B | 32795 | Bmulti locked for reconfiguration. |
| 801C | 32796 | Entry error. |
| 801D | 32797 | Station Active. |

# INDEX

## Documentation Evaluation Form

Title: B 20 Systems BMULTI Reference Manual 5.0 Form No: 1182284

Date: April, 1985

Burroughs Corporation is interested in receiving your comments and suggestions regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Comment/Suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error ☐ Other

Comments:

From:

Name _____

Title _____

Company _____

Address _____

_____

Phone Number _____ Date _____

Remove form and mail to:

Burroughs Corporation
Corporate Product
Information East
209 W. Lancaster Ave.
Paoli, PA 19301 U.S.A.