1.0 INTRODUCTION

The Microdata Reality Binary Synchronous Communication Process is an emulation of the IBM 2780 Data Transmission Terminal. The Binary Synchronous Communications Process (Bisync) can be used to transmit Reality file items and files at line speed to another Reality or any other machine following the IBM 2780 method of bisynchrouous communicatio protocol. Supported special features include transparency, multiple record blocks, EBCDIC transmission, ASCII transmiss in transparent text, extended retry feature, printer horizontal format control, auto answer, and auto turnaround.

2.0 OPERATOR'S MANUAL

The Reality Bisynchrounous Communciation Process may be assigned to any line (terminal) in the system and controlled from any or all the remaining terminals. When the bisync process is assigned to a line, the terminal attached to that line is dedicated to the process and may not be used for any other purpose until the bisync process has been deactivated. Data may be readied for transmission either before or after the bisync process has been activated. The data to be transmitted is stored in items in Reality files. As many items as desired or a whole file may be structured into a transmission message. When activated, the current status of the bisync process is displayed on terminal to which the process is attached. Such things as whether or not the telephone line is connected, or if a message is currently being transmitted or received are displayed. The bisync process is controlled by several verbs which control the process' modes of operation, the structuring of transmission messages, and the filing of received messages.

There are ten verbs used to control the bisync process:

TRANSMIT, DISPLAY-MSG-QUE, START-BSC, STOP-BSC,
DISPLAY-MESSAGE, SPOOL-MESSAGE, FILE-MESSAGE, DUMMY-MESSAGE
KILL-MESSAGE, ABORT-BSC, and RESTART-BSC.

Following is a description of the verbs and their options.

2.1 TRANSMIT

The TRANSMIT verb will structure a transmission message from items within a Reality file. Each attribute within an item will become a record in the message. Each record is limited to 80 characters, therefore, if any attribute is greater than 80 characters, the excess will be truncated and lost. The general verb format is:

```
TRANSMIT file-name item-list (options)
```

The item-list may consist of one or more items within the file, separated by blanks, or all the items in the file may be specified by using an asterisk (*). The valid options are:

2.1.1 <u>N -transmit normal EBCDIC</u>

The ASCII data within the specified items is converted to EBCDIC and structured into a transmission message in the normal text mode of bisync.

2.1.2 <u>T - transmit transparent ASCII</u>

The unconverted ASCII data within the specified items is structured into a transmission message in the transparent text mode of bisync. This option should also be used whenever transmitting non-ASCII data such as Reality object code. Records are padded with ASCII blanks to 80 characters.

2.1.3 C - transmit transparent EBCDIC

The ASCII data within the specified items is converted to EBCDIC and structured into a transmission message in the transparent text mode of bisync. Short records are padded with EBCDIC blanks to 80 characters.

2.1.4 S - transmit short records

Records are limited to a maximum of 80 characters. Attributes in items which have less than 80 characters (0-79) have an EM character (hex 19) appended to them and are transmitted as less than eighty character records. This mode is recommended when possible since it will reduce transmission time.

2.1.5 F - transmit fixed length records

Records are 80 characters long. Attributes shorter than 80 characters are padded with blanks until eighty characters long. Attributes longer than eighty characters are truncated.

2.1.6 X - transmit two records per block

Each transmission block, normal or transparent text, will contain two, short or fixed length records.

2.1.7 <u>M - transmit multiple records per block (1-7)</u>

Transmission blocks are limited to 400 characters including line control characters. In this mode, records are placed in transmission blocks until either seven records are in the block or the block size exceeds 400 characters. Because of the 400 character limitation, when fixed length or transparent records are being transmitted, only four records will fit in a block.

2.1.8 Default values for the options are (N,S,M). Therefore, if the only option specified is (F), for example, the transmission message will be structured in normal text (N option), fixed length records (F option), and with multiple records per block (M option).

2.1.9 Each message formed by the TRANSMIT verb will assign a four digit identification number in the range 0000 to 9999. When activated the bisync process transmits messages with the lowest identification number first. After structuring a transmission message the message:

```
MESSAGE 'XXXX' ENTERED IN QUEUE,
```

where 'XXXX' is the identification number assigned to the message, is returned to the operator.

2.1.10 An example of a TRANSMIT command would be:

```
:TRANSMIT ACCOUNTS 10985 21662 (C)
MESSAGE '0154' ENTERED IN QUEUE
```

Items 10985 and 21662 from the ACCOUNTS file are structured into a transparent text EBCDIC message which has been assigned the identification number 0154.

2.2 DISPLAY-MSG-QUE

The DISPLAY-MSG-QUE verb displays the identification number and some other parameters of messages that are waiting to be transmitted or that have been received. The general verb format is:

```
DISPLAY-MSG-QUE (options)
```

2.2.1 <u>P - print message queue on printer</u>

This is the only valid option for the DISPLAY-MSG-QUE verb.

2.2.2 The message queue is displayed in the following format:

```
name account channel time date status
```

name - the identification number assigned to the message by the system. The identification number is in the range 0000-9999.

account - the account onto which the operator who initiated the message transmission was logged.

channel - the line number of the terminal from which the message's transmission was initiated.

time - the time when the message was entered into the queue, either the time the message was put into the queue with the TRANSMIT verb or the time the message was received.

date - the date on which the message was put into the queue.

status - the current status of the message which can be:

T	- the message is waiting to be transmitted
T*	- the message is currently being transmitted
R	- the message has been received
R*	- the message is currently being received
TH	- the message was partially transmitted when a condition requiring operator intervention arose
RH	- the message was partially received when a condition requiring operator intervention arose

2.2.3 An example of the DISPLAY-MSG-QUE verb is:

```
:DISPLAY-MSG-QUE
BSC MESSAGE QUEUE
NAME  ACCOUNT  CHANNEL    TIME        DATE       STATUS
0154  SYSPROG     0      9:46:05   9 MAY 1975     T*
0155                     9:51:16   9 MAY 1975     R
0156  ALICE       5     10:01:47   9 MAY 1975     T
END OF QUEUE
```

Message 0154 which was originated from the terminal attached to line zero and logged onto the SYSPROG account at 9:46:05 on 9 MAY 1975 is currently being transmitted. Message 0155 was received at 9:51:16 on 9 MAY 1975; there is no account name or channel number associated with a received message. Message 0156 was entered in the queue by ALICE from line 5 at 10:01:47 on 9 MAY 1975 and is waiting to be transmitted.

2.3 DUMP-MESSAGE

The DUMP-MESSAGE verb is used to display the structured message in hex. The general verb format is:

```
DUMP-MESSAGE message-number (options)
```

The message number is the identification number assigned to the message.

2.3.1 <u>P - display message on the printer</u>

The P option is the only valid option.

2.3.2 An example of the DUMP-MESSAGE verb is:

```
:DUMP-MESSAGE 0025
MESSAGE '0025'
3232
02C1C2C3C4191F
3232
02C5C6C7C81903
```

Message 0025 is dumped to the terminal in hex, including all line control characters.

2.4 SPOOL-MESSAGE

The SPOOL-MESSAGE verb is used to list a message on the printer, one record to the line. The general format of the verb is:

```
SPOOL-MESSAGE message-number (options)
```

The message-number is the identification number assigned to the message by the system. The valid options are:

2.4.1 S - suppress tabs and forms control, and allow automatic paging.

This option should be used if the received message does not contain imbedded forms control characters. The message will then be automatically paged.

2.4.2 H - hold message in message queue

The message is printed, but is not subsequently deleted from the message queue, otherwise the message is automatically deleted. This option may be used for printing multiple copies of the message or for printing and filing the same message.

2.4.3 C - convert EBCDIC to ASCII when printing a transparent text message.

This option will normally be specified when printing transparent text.

2.4.4 An example of the SPOOL-MESSAGE verb is:

```
:SPOOL-MESSAGE 0256 (S,C)
MESSAGE '0256' DELETED
```

Message 0256 a transparent text message was printed after converting the EBCDIC text to ASCII; the message was automatically paged while printing; since the H option was not specified, the message was deleted from the queue after printing.

2.5 FILE-MESSAGE

The FILE-MESSAGE verb is used to place a message in a file. The normal mode of operation is for each record in the message to become a different item in the file, each record being assigned a sequential item-id starting with 0001. The message may also be assembled into one item, each record becoming an attribute of the item; the filed message must not exceed 32K bytes, the maximum item size. The general format of the verb is:

```
FILE-MESSAGE message-name (options)
TO: file-name [item-name]
```

The message-name is the identification number assigned to the message. After typing in the first line, the operator is prompted with TO:, to which he replies with the file-name of the file into which the message is to be placed, and with the item-name which he wishes assigned to the message if the item option is specified. The valid options for the verb are:

2.5.1 <u>T - process horizontal tabs</u>

If the message contains a tab format record as its first record and the message records contain horizontal tab characters, tabbing will be done on the records as they are filed by inserting an appropriate number of blanks between fields in the record. If not specified, the tab characters will be left in the records.

2.5.2 <u>F - process vertical forms control</u>

If the records in the message begin with forms control characters, the appropriate form and line feeds are inserted in the filed message, if the F option is specified. If not specified, the forms control characters are left at the beginning of the filed records.

2.5.3 I - file into one item

When the I option is specified, the message is placed in one item in the file, each record of the message becoming an attribute of the item. The records of the message must not exceed a total of 32K bytes, the maximum item size, If the message is too long, an error message is given to the operator, the item being built is deleted fron the file, and the message remains in the message queue. This option will be used when filing received Reality object code. If this option is not specified the records in the message will be filed one to an item within the file. The records will be given a numeric item-id as they are filed, the first record becoming item 0001, the second record item 0002, and so on.

2.5.4 R - format Reality object code into file item

The R option is used when a frame of Reality object code has been received as a message. The I option (file message into one item) is set whenever the R option is specified. The Reality object code will be properly formatted into an item so that it may be loaded into the system.

2.5.5 C - convert EBCDIC to ASCII when filing transparent text

The EBCDIC data in the received message is converted to ASCII before the data is filed. This option should normally be specified unless it is known that the message consists of ASCII data from another Reality system or hexadecimal data.

2.5.6 Two examples of the FILE-MESSAGE verb are:

```
:FILE-MESSAGE 1846 (T,F,C)
TO: INVOICES
```

The message 1846 is placed into the file INVOICES. Tabbing, forms control, and EBCDIC to ASCII conversion are specified as options.

```
:FILE-MESSAGE 0006 (I)
TO: ADDRESS-LIST BOSTON
```

The message 0006 is placed in the item BOSTON in the ADDRESS-LIST file.

2.6 START-BSC

The START-BSC verb is used to activate the bisync process. The process is assigned to a line by the verb other than the line from which the verb is entered. The bisync process may not be active on more than one line at a time. The line and terminal connected to that line become dedicated to the bisync process and may be used for no other purpose while the process is activated. The general format of the verb is:

```
START-BSC line-number (options)
```

The line number is the line to which the bisync process will be assigned. The valid options are:

2.6.1 <u>P - primary station</u>

The Reality system in which the bisync process is operating is designated the primary station in a communication link.

2.6.2 S - secondary station

The Reality system in which the bisync process is operating is designated the secondary station in a communications link.

2.6.3 A - attended mode

The attended option is set when operator intervention is desired during an abnormal condition that occurs while receiveing or transmitting a message. The normal mode of operation for the bisync process is that if an abort condition (such as line failure) occurs while receiving, the partially received message is deleted from the message queue and that if an abort condition occurs while transmitting, the retransmission of the message is from the beginning of the message. In the attended mode, when an abort condition is encountered, the operator must restart the bisync process, at which item, he may select to either restart the transmission or reception of a message from where the message was interrupted, or restart the reception or transmission from the beginning. Operator intervention is required, therefore, this option should not be specified when using the process in an unattended, autoanswer mode. The RESTART verb is used for restarting the bisync process.

2.6.4 An example of the START-BSC verb is:

```
:START-BSC 5 (S)
BISYNC PROCESSOR INITIATED
```

The bisync process is activated on line five as a secondary station. The terminal attached to line five will also display the message:

```
    BISYNC PROCESSOR INITIATED
```

If the bisync process has been previously started, the message:

```
    BISYNC PROCESSOR ALREADY ACTIVE
```

is returned to the operator.

2.7 STOP-BSC

The STOP-BSC verb isused to inactivate the bisync process after reception of an incoming message is finished, or the transmission of all outstanding messages in the message queue is finished. There are no options for the verb. An example of the verb is:

```
    :STOP-BSC
    BISYNC PROCESSOR TERMINATED
```

The bisync process is stopped after completion of the present operation and the termination message is returned to the operator. If bisync is already inactive, the message:

```
    BISYNC PROCESSOR INACTIVE
```

is returned to the operator.

2.8 ABORT-BSC

The ABORT-BSC verb is used to immediately deactivate the bisync process. If the process is currently receiving a message, further reception is stopped, and the already received portion of the message is deleted. If the process is currently transmitting a message, the transmission is stopped, and the message is returned to the message queue. An example of the verb is:

```
:ABORT-BSC
BISYNC PROCESSOR TERMINATED
```

The bisync process is immediately stopped and the termination message is returned to the operator. If the bisync process is already inactive, the message:

```
BISYNC PROCESS INACTIVE
```

is returned to the operator.

2.9 RESTART-BSC

The RESTART-BSC verb is used to restart the bisync process after an abort condition has occurred while receiving or transmitting a message when the A, attended, option has been specified in the START-BSC verb. If the bisync process is restarted transmitting a message, the block of data that was last attempted to be transmitted is sent as the first block of the new transmission. If the bisync process is restarted receiving a message, the first block of data received is appended to the end of the aborted partially received message. This procedure may lead to duplication of several lines of data at the point that the message interruption occurred, but usually not. The general format for the verb is:

[illegible]

```
RESTART-BSC (options)
```

The only option for the verb is:

2.9.1 N - new start

If the N option is specified the partially received message is deleted and the next reception starts a new message, or the partially transmitted message is restarted from the beginning on the next transmission. If the option is not specified, the transmission or reception of the aborted message is restarted as outlined above.

2.9.2 An example of the verb is:

```
:RESTART-BSC
BISYNC PROCESS RESTARTED
```

The bisync process is restarted with the message that was being processed when the abort occurred, and the restart message is returned to the operator. If the bisync process was inactive the message:

```
BISYNC PROCESS INACTIVE
```

is returned to the operator.

3.0 GENERAL CONSIDERATIONS ON BINARY SYNCHRONOUS COMMUNICATIONS

The general principles of binary synchronous communications as implemented in the Reality bisync process will be considered. A more detailed discussion may be obtained in the IBM publications "General Information - Binary Synchronous Communications" (GA27-3004) and "Component Description: IBM 2780 Data Transmission Terminal" (GA27-3005).

3.1 DATA LINK CONTROL CHARACTERS

Following is a list of the data link control characters, their hex equivalent in EBCDIC, and their function.

3.1.1 SYN (32) - Synchronous Idle

The SYN character is used to establish and maintain synchronization on the data link. Each transmission must be preceded by at least two SYN characters.

3.1.2 SOH (01) - Start of Heading

The SOH character precedes a block of heading characters. The Reality bisync process will treat an SOH character as an STX character.

3.1.3 STX (02) - Start of Text

The STX character precedes a block of data characters.

3.1.4 ETB (26) - End of Transmission Block

The ETB character indicates the end of a block of characters started with an STX. ETB indicates that more data is to follow after the block is acknowledged by the receiving terminal.

3.1.5 ITB (1F) - End of Intermediate Transmission Block

The ITB character ends a portion of a block of data. When more than one record is sent in a block, the first record starts with an STX and ends with an ITB; each subsequent record ends with an ITB, except the last which ends with an ETB or ETX.

3.1.6 ETX (03) - End of Text

The ETX character ends the last block of data in a message. It indicates that the records in the first block of data through the block ended by the ETX form one complete message. Unless this character is sent before an EOT character, the Reality bisync process will assume the message was interrupted and indicate an error condition.

3.1.7 EOT (37) - End of Transmission

The EOT character is sent by a terminal to indicate that it has no more messages to transmit.

3.1.8 ENQ (2D) - Enquiry

The ENQ character is used in bidding for the line when the terminal has data to transmit, or is used to request a retransmission of the last acknowledgement to a transmitted block of data.

3.1.9 ACKO/ACKI (1070/1061) - Positive Acknowledgement

The ACK characters, ACKO or ACKI, are used to indicate that the last block of data was correctly received. The responses ACKO and ACKI are alternated for each block of data; ACKO is used as a response to a line bid (ENQ).

3.1.10 WACK (106B) - Wait Before Transmit Positive Acknowledgement

The WACK character is used as a positive acknowledgement to a block of data, but indicates that the receiving station is not yet ready for another block of data. The Reality bisync process will accept, but not transmit the WACK character.

3.1.11 NAK (3D) - Negative Acknowledgement

The NAK character is used to indicate that an error was detected while receiving the last block of data and requests a retransmission of that block.

3.1.12 DLE (10) - Data Link Escape

The DLE character is used as a supplimentary data link control character as in ACKO or WACK, or in conjunction with the block delimiting character; for example, STX and ITB, to indicate the transparent mode of operation.

3.1.13 RVI (106B) - Reverse Interrupt

The RVI character is used as a positive acknowlegement to a block of data and requests the transmitting station to stop further transmission and go into the receive mode. The Reality bisync process will accept, but not transmit the RVI character.

3.1.14 DLE EOT (1037) - Disconnect

The DLE EOT character is sent by a terminal on a switched (dial-up) line to indicate that it is "hanging-up" the line.

3.2 PRIMARY AND SECONDARY STATIONS

When a bisync station is initiated, it is designated as either a primary or secondary station. When a primary station is bidding for a line, it will send out ENQ characters every second, waiting one second for the other terminal to reply. When a secondary station is bidding for a line, it will send out ENQ characters every three seconds, waiting three seconds for the other terminal to reply. This has a net effect of giving the primary station priority over secondary stations on the line when it has a message to transmit.

3.3 BLOCK CHECK CHARACTER (BCC)

Binary synchronous communications use a cyclic redundancy checking as a means of detecting transmission errors in data. The message block that is to be transmitted is considered as one long binary number which is divided by a known 16 bit binary number. This division produces a 16 bit remainder which is appended to the transmission block. The receiving station also divides the received block by the known 16 bit number. If the remainder it generates is the same as the remainder it received, the block is assumed to have been received correctly. The polynomial used as divisor is:

$$X^{16} + x^{15} + x^{2} + 1$$

which corresponds to the binary number

    1100000000000101

3.4 BLOCK STRUCTURE

Blocks will have two basic formats depending upon whether they are normal text blocks or transparent text blocks.

3.4.1 Normal Text Blocks

Normal text blocks have four basic formats:

a) SYN SYN STX text ETX BCC
b) SYN SYN STX text ETB BCC
c) SYN SYN STX text ITB BCC SYN SYN STX text ETX BCC
d) SYN SYN STX text ITB BCC text ETX BCC

Block c or d are used for multiple records per block.

3.4.2 Transparent Text Blocks

a) SYN SYN DLE-STX text DLE-ETX BCC
b) SYN SYN DLE-STX text DLE-ETB BCC
c) SYN SYN DLE-STX text DLE-ITB BCC SYN SYN DLE-STX text DLE-ETB BCC

Transparent text mode is used when the data may contain other than the normal ASCII character set, such as when transmitting Reality object code, or when transmitting packed data. Transparent mode is initiated by a DLE-STX and is terminated by DLE-ITB, DLE-ETB or DLE-ETX. If a DLE character appears in the data, it is preceded by a second DLE character, indicating that the first DLE is data which is stripped out on the receiving end.

This precludes data such as DLE-ETX being recognized as a control character instead of data.

3.5 EXAMPLES OF TRANSMISSION PROCEDURES

3.5.1 Bidding For The Line

```
CPU 1                CPU 2

ENQ →              ← ENQ
:
1 sec
:
ENQ →
                   ← ACKO
```

CPU 1 is the primary station and CPU 2 is the secondary station. Both CPU's simultaneously bid for the line. Since CPU 1 is the primary station, it sent a second ENQ character after one second. CPU 2, waiting three seconds for the reply to its ENQ, receives CPU 1's ENQ, replies with an ACKO, and goes into the receive mode.

3.5.2 Message Transmission

```
CPU 1                CPU 2

ENQ →
                   ← ACKO
STX text ETB →
                   ← ACK1
STX text ETX →
                   ← ACKO
EOT →
:
21 Sec Timeout
:
DLE-EOT →
```

CPU 1 bids for the line and receives an acknowledgement from CPU 2. CPU 1 sends two blocks of data to CPU 2, receiving the correct positive acknowledgements from CPU 2. CPU 1 has nothing else to send, so it sends an end of transmission character, and goes into the receive mode, waiting for CPU 2 to send any data it may have. CPU 2 has nothing to send, so CPU 1 waits 21 seconds and sends a disconnect, DLE EOT, and hangs up the line.

3.5.3 Retransmission Examples

```
      CPU 1                CPU 2

 1
      ENQ →
                         ← ACKO
      ENQ →
 2                       ← ACKO
      STX text ETB →
                         ← NAK
      STX text ETB →
 3                       ← ACK1
      STX text ETB →
      .
      .
      3 Sec Timeout
      .
      .
      ENQ →
                         ← ACK1
      STX text ETB →
 4                       ← ACKO
      STX text ETX →
                         ← ACK1
```

[1] CPU 1, bidding for the line, missed CPU 2's acknowledgement, so after timing out; it repeats its linebid. [2] After receiving CPU 2's acknowledgement, CPU 1 sends the first block of data. CPU 2 detected an error in the block and sends back a NAK to which CPU 1 replies with a retransmission of the block. [3] After receiving CPU 2's acknowledgement, CPU 1 sends the second block of data. CPU 1, after waiting three seconds without receiving a response, sends an ENQ; requesting CPU 2 to repeat its last response. CPU 2's response of an out of sequence ACK indicates that CPU 2 did not receive the last block so CPU 1 retransmits the block. [4] After receiving the correct response, CPU 1 sends the final block of data.

3.5.4 Use of RVI

CPU 1	CPU 2
ENQ →	
	← ACKO
STX text ETX →	
	← RVI
EOT →	
	← ENQ
ACKO →	
	← STX text ETX
ACK1 →	
	← EOT
DLE EOT →	

CPU 1 wins the line bid and sends one block of data. CPU 2 sends back an RVI as a positive acknowledgement, indicating that it has a message to send. CPU 1 sends back an EOT and acknowledges CPU 2's line bid. Transmission then continues in the normal manner.

4.1 TRANSMITTING A REALITY FILE

```
1
:START-BSC 1
01 NOV 1975 13:49:09 BISYNC PROCESSOR INITIATED

3
:TRANSMIT ACCOUNTS *
1267
54862
7914
7902
980002
MESSAGE '0001' ENTERED IN QUEUE

7
:STOP-BSC
01 NOV 1975 14:02:15 BISYNC PROCESSOR TERMINATED
```

---

[1]The bisync processor was started on line one in the unattended mode of operation. [2]The initiation message was returned on the terminal connected to line one. [3]All the items (five of them) in the ACCOUNTS file were structured into a message and entered in the message queue as message 0001. [4]The operator called the computer which will receive the message and placed the data phone in DATA after the connection had been made. The communications line enabled message was returned to the bisync terminal.

```
2
:01 NOV 1975 13:49:09 BISYNC PROCESSOR INITIATED

4
01 NOV 1975 14:01:54 BISYNC COMMUNICATIONS LINE ENABLED

5
01 NOV 1975 14:02:04 0001 SYSPROG 00 13:50:05 01 NOV 1975 T*

6
01 NOV 1975 14:03:21 11 BLOCKS TRANSMITTED
01 NOV 1975 14:03:21 11 BLOCKS RECEIVED
01 NOV 1975 14:03:21 0 TRANSMISSION ERRORS
01 NOV 1975 14:03:22 MESSAGE TRANSMISSION COMPLETE
01 NOV 1975 14:03:23 MESSAGE '0001' DELETED

8
01 NOV 1975 14:03:23 BISYNC COMMUNICATIONS LINE DISCONNECTED
BISYNC PROCESSOR INACTIVE

LOGON PLEASE:
```

---

[5]After transmission of message 0001 had begun, the transmission message was returned to the bisync terminal. [6]After completion of the message transmission, the transmission statistics were returned to the bisync terminal. The number of blocks transmitted and received, and the number of retransmissions were listed. Message 0001 was then deleted from the message queue. [7]While the message was being transmitted, the operator entered the STOP-BSC verb, terminating the bisync process at the end of the job. [8]The bisync process, seeing that the STOP-BSC verb had been executed, disconnected the line and deactivated itself after transmitting message 0001.

4.2 RECEIVING A REALITY FILE

```
1

:START-BSC 1
01 NOV 1975 14:55:02 BISYNC PROCESSOR INITIATED

6

:STOP-BSC
01 NOV 1975 15:16:23 BISYNC PROCESSOR TERMINATED

8

:FILE-MESSAGE 0002
TO: RECEIVED-INVOICES
MESSAGE '0002' DELETED
```

---

[1]The bisync process was started on line one in the unattended mode. [2]The initiation message was returned on the bisync process' terminal. [3]Assuming that the data phone was in AUTO, several minutes later a call was received and the communication line enabled. [4]Message 0002 was received. There were 113 blocks received with two retransmissions. [5]After waiting 21 seconds without receiving anything else, the bisync process disconnected the line. [6]The operator disconnected the line using the STOP-BSC verb. [7]The STOP-BSC verb cleared wait for a call state in the bisync process. The process saw that the line was disconnected and that the stop command had been issued, so the process deactivated itself, and returned to logon.

```
2
:01 NOV 1975 14:55:02 BISYNC PROCESSOR INITIATED

3
01 NOV 1975 15:01:36 BISYNC COMMUNICATIONS LINE ENABLED

4
01 NOV 1975 15:01:42 0002          15:01:42 01 NOV 1975 R*
01 NOV 1975 15:07:16 112 BLOCKS TRANSMITTED
01 NOV 1975 15:07:16 113 BLOCKS RECEIVED
01 NOV 1975 15:07:16 2 TRANSMISSION ERRORS
01 NOV 1975 15:07:17 MESSAGE TRANSMISSION COMPLETE

5
01 NOV 1975 15:07:38 BISYNC COMMUNICATIONS LINE DISCONNECTED

7
01 NOV 1975 15:16:23 BISYNC COMMUNICATIONS LINE ENABLED
01 NOV 1975 15:16:23 BISYNC COMMUNICATIONS LINE DISCONNECTED

BISYNC PROCESSOR INACTIVE

LOGON PLEASE:
```

---

[8]The operator then filed the received message in the RECEIVED-INVOICES file.

4.3 TRANSMITTING REALITY OBJECT CODE

```
1

:STRIP SYSTEM-MODES DBI
DESTINATION: HOLD

2

TRANSMIT HOLD DBI (T)
MESSAGE '0003' ENTERED IN QUEUE

3

:START-BSC 1
01 NOV 1975 15:30:00 BISYNC PROCESSOR INITIATED
```

---

[1]The STRIP verb was used to strip off the source code from the frame of code, DBI. When transmitting Reality code, the operator should insure that the first five lines of the frame are comments to preclude the possibility of source code being lost when the lines of code are truncated to 80 characters, since the STRIP verb does not affect the first five lines of code in the frame. [2]The stripped frame of code was then structured into a transparent text transmission message. [3]The operator then started the bisync process with the START-BSC verb.

[4]

```
:01 NOV 1975 15:30:00 BISYNC PROCESSOR INITIATED
```

[5]

```
01 NOV 1975 15:32:56 BISYNC COMMUNICATION  LINE ENABLED
01 NOV 1975 15:33:01 0003 SYNPROG 00 15:25:13 01 NOV 1975 T*
01 NOV 1975 15:36:50 52 BLOCKS TRANSMITTED
01 NOV 1975 15:36:50 52 BLOCKS RECEIVED
01 NOV 1975 15:36:50 0 TRANSMISSION ERRORS
01 NOV 1975 15:36:51 MESSAGE TRANSMISSION COMPLETE
```

[6]

```
01 NOV 1975 15:37:12 BISYNC COMMUNICATIONS LINE DISCONNECTED
```

---

[4]The initialization message was returned to the bisync process terminal. [5]After the line connection was made, the frame of code was transmitted. [6]After transmitting its message, the bisync process went into the receive state. Since nothing was received within 21 seconds, the process disconnected the line.

4.4 RECEIVING REALITY OBJECT CODE

```
1

:START-BSC 1
01 NOV 1975 15:52:26 BISYNC PROCESSOR INITIATED

4

:ED SYSTEM-MODE DBI
.TOP
.FD
DBI DELETED

5

:FILE-MESSAGE 0004 (R)
TO: SYSTEM-MODES DBI
```

---

[1]The bisync process was started on line one by the operator. [2]The initialization message was returned to the bisync process' terminal. [3]After the line connection was made, message 0004 was received. [4]Since the operator knew the frame DBI was recieved, he first deletes the old DBI from the SYSTEM-MODES file using the EDITOR. [5]The message was then filed into SYSTEM-MODES DBI using the R option (file Reality object code) of the FILE-MESSAGE verb.

2

```
:01 NOV 1975 15:56:26 BISYNC PROCESSOR INITIATED
```

3

```
01 NOV 1975 15:59:30 BISYNC COMMUNICATIONS LINE ENABLED
01 NOV 1975 15:59:45 0004    15:59:45 01 NOV 1875 R*
01 NOV 1975 16:05:02 52 BLOCKS TRANSMITTED
01 NOV 1975 16:05:02 53 BLOCKS RECEIVED
01 NOV 1975 16:05:03 0 TRANSMISSION ERRORS
01 NOV 1975 16:05:03 MESSAGE TRANSMISSION COMPLETE
01 NOV 1975 16:05:25 BISYNC COMMUNICATIONS LINE DISCONNECTED
```

4.5 TRANSMITTING A DITTO JOB TO AN IBM 360 OPERATING UNDER DOS/POWER

Assume there is a file called JOB-CONTROL and this file contains two items LOGON and LOGOFF.

<u>LOGON</u>

```
001 * .. RJSTART MICRO
002 * .. LOGON MICRO
003 * $$ JOB DITTO
004 * $$ PRT H
005 * $$ PUN H
006 // JOB DITTO
007 // UPSI 1
008 // EXEC DITTO
009 $$DITTO CPU
```

<u>LOGOFF</u>

```
001 /*
002 $$DITTO EOJ
003 * $$ EOJ
004 * .. LOGOFF
005 * .. RJEND
```

```
1

:START-BSC 1
01 NOV 1975 16:20:02 BISYNC PROCESSOR INITIATED

3

:TRANSMIT JOB-CONTROL LOGON (F)
MESSAGE '0005' ENTERED IN QUEUE

4

:TRANSMIT DATA * (F)
1034
92685
:
28
4678
MESSAGE '0006' ENTERED IN QUEUE

5

:TRANSMIT JOB-CONTROL LOGOFF (F)
MESSAGE '0007' ENTERED IN QUEUE
```

---

[1]The operator started the bisync process on line one and [2]the initialization message was returned on the bisync process' terminal. [3]The logon job control cards were entered in the message queue. [4]All the items in the DATA file, the data which is to be dittoed, was entered in the message queue. [5]The logoff job control cards were entered in the message queue. [6]The job was sent to the IBM360. The operator disconnected the line, waiting for the job to be run.

```
2

:01 NOV 1975 16:20:02 BISYNC PROCESSOR INITIATED

6

01 NOV 1975 16:27:15 BISYNC COMMUNICATIONS LINE ENABLED
01 NOV 1975 16:27:20 0005 SYSPROG 00 16:20:56 01 NOV 1975 T*
01 NOV 1975 16:28:32 3 BLOCKS TRANSMITTED
01 NOV 1975 16:28:32 3 BLOCKS RECEIVED
01 NOV 1975 16:28:32 0 TRANSMISSION ERRORS
01 NOV 1975 16:28:33 MESSAGE TRANSMISSION COMPLETE
01 NOV 1975 16:28:35 MESSAGE '0005' DELETED
01 NOV 1975 16:28:40 0006 SYSPROG 00 16:23:10 01 NOV 1975 T*
01 NOV 1975 16:40:16 242 BLOCKS TRANSMITTED
01 NOV 1975 16:40:16 243 BLOCKS RECEIVED
01 NOV 1975 16:40:16 12 TRANSMISSION ERRORS
01 NOV 1975 16:40:17 MESSAGE TRANSMISSION COMPLETE
01 NOV 1975 16:40:26 MESSAGE '0006' DELETED
01 NOV 1975 16:40:31 0007 SYSPROG 00 16:28:14 01 NOV 1975 T*
01 NOV 1975 16:41:02 2 BLOCKS TRANSMITTED
01 NOV 1975 16:41:02 2 BLOCKS RECEIVED
01 NOV 1975 16:41:02 0 TRANSMISSION ERRORS
01 NOV 1975 16:41:03 MESSAGE TRANSMISSION COMPLETE
01 NOV 1975 16:41:05 MESSAGE '0007' DELETED
01 NOV 1975 16:41:26 BISYNC COMMUNICATION LINE DISCONNECTED
```

```
1

:START-BSC 1
01 NOV 1975 17:35:53 BISYNC PROCESSOR INITIATED

3

:TRANSMIT JOB-CONTROL OUTPUT (F)
MESSAGE '0008' ENTERED IN QUEUE

6

:TRANSMIT JOB-CONTROL LOGOFF (F)
MESSAGE '0010' ENTERED IN QUEUE

8

:FILE-MESSAGE 0009
TO: ACCOUNTS
```

---

[1]The operator started the bisync process on line one, and [2]the initialization message was returned on the bisync process' terminal. [3]The operator there entered the output job control cards in the message queue. [4]The line connection was then made and the output cards sent to the 360. [5]Upon receiving the output cards, the 360 transmitted the results of the previous ditto job. [6]While receiving the ditto job, the operator entered the logoff cards in the message queue. This must be done while receiving the output, otherwise, if the logoff is placed in the queue at the same time as the logon, the logoff cards would be transmitted before receiving the output, aborting the job on the 360.

```
2

:01 NOV 1975 17:35:53 BISYNC PROCESSOR INITIATED

4

01 NOV 1975 17:38:10 BISYNC COMMUNICATIONS LINE ENABLED
01 NOV 1975 17:38:15 0008 SYSPROG 00 17:36:15 01 NOV 1975 T*
01 NOV 1975 17:38:58 1 BLOCKS TRANSMITTED
01 NOV 1975 17:38:58 1 BLOCKS RECEIVED
01 NOV 1975 0 TRANSMISSION ERRORS
01 NOV 1975 17:38:58 MESSAGE TRANSMISSION COMPLETE
01 NOV 1975 17:39:00 MESSAGE '0008' DELETED

5

01 NOV 1975 17:39:07 0009     17:39:07 01 NOV 1975 R*
01 NOV 1975 17:51:36 236 BLOCKS TRANSMITTED
01 NOV 1975 17:51:36 237 BLOCKS RECEIVED
01 NOV L(&% 17:51:37 6 TRANSMISSION ERRORS
01 NOV 1975 17:51:38 MESSAGE TRANSMISSION COMPLETE

7

01 NOV 1975 17:51:42 0010 SYSPROG 00 17:42:26 01 NOV 1975 T*
01 NOV 1975 17:52:03 1 BLOCKS TRANSMITTED
01 NOV 1975 17:52:03 1 BLOCKS RECEIVED
01 NOV 1975 17:52:03 0 TRANSMISSION ERRORS
01 NOV 1975 17:52:03 MESSAGE TRANSMISSION COMPLETE
01 NOV 1975 17:52:05 MESSAGE '0010' DELETED
01 NOV 1975 17:52:26 BISYNC COMMUNICATIONS LINE DISCONNECTED
```

---

[7]The logoff cards were then transmitted after receiving the output. [8]The operator then filed the received message in the ACCOUNTS file.