1989-12

Turbo Pascal implementation of a distributed processing network of MS-DOS microcomputers connected in a master-slave configuration

Ard, Nelson C.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TURBO PASCAL IMPLEMENTATION OF A DISTRIBUTED PROCESSING NETWORK OF MS-DOS MICROCOMPUTERS CONNECTED IN A MASTER-SLAVE CONFIGURATION

by

NELSON C. ARD

DECEMBER 1989

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS Unrestricted
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 52	7a NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
TURBO PASCAL IMPLEMENTATION OF A DISTRIBUTED PROCESSING NETWORK OF MS-DOS MICROCOMPUTERS CONNECTED IN A MASTER-SLAVE CONFIGURATION

12 PERSONAL AUTHOR(S)
Ard, Nelson C.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	December 1989	308

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Distributed Processing, Local Area Network, Star Network, Turbo Pascal

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis describes the design and implementation of a distributed processing network of IBM PC compatible computers capable of performing parallel processing tasks. The network is a star cluster local area network, with the central computer controlling the operations of the satellite computers on a sequential basis.

The local area network software operates over the computer's standard RS-232C communications ports, and is currently implemented to allow the central computer to operate two satellite computers. Processing tasks are dispatched to the satellite computers as programs which run to completion on the satellite computers. Utility programs within the software include file and message transfer to start the programs on the satellite computers and to obtain the output of the remotely executed program, configuration utilities to set the communications port parameters, and windowing utilities for display of information normally presented on the remote computer's display. The program is implemented in Turbo Pascal 4.0 under the MS-DOS operating system, version 3.21.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Professor Uno Kodres	22b TELEPHONE (Include Area Code) (408) 646-2197 22c OFFICE SYMBOL 52Kr

Approved for public release; distribution is unlimited.

Turbo Pascal Implementation of a Distributed Processing Network of MS-DOS Microcomputers Connected in a Master-Slave Configuration

by

Nelson C. Ard
B.S., Virginia Polytechnic Institute and State University, 1974

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
DECEMBER 1989

ABSTRACT

This thesis describes the design and implementation of a distributed processing network of IBM PC compatible computers capable of performing parallel processing tasks. The network is a star cluster local area network, with the central computer controlling the operations of the satellite computers on a sequential basis.

The local area network software operates over the computer's standard RS-232C communications ports, and is currently implemented to allow the central computer to operate two satellite computers. Processing tasks are dispatched to the satellite computers as programs which run to completion on the satellite computers. Utility programs within the software include file and message transfer to start the programs on the satellite computers and to obtain the output of the remotely executed program, configuration utilities to set the communications port parameters, and windowing utilities for display of information normally presented on the remote computer's display. The program is implemented in Turbo Pascal 4.0 under the MS-DOS operating system, version 3.21.

DISCLAIMER

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Several of the terms used in this thesis refer to commercial products for which the manufacturer or vendor holds a trademark. All registered trademarks appearing in this thesis are cited below with the firm holding the trademark in lieu of citing the holder with each individual occurance of the trademark.

Bell Laboratories, Murray Hill, New Jersey
UNIX Operating System

Board of Regents, University of California at San Diego (USCD), San Diego, California
UCSD Pascal Programming Language

Borland International, Incorporated, Scotts Valley, California
Turbo Pascal Programming Language

Digital Research Incorporated, Pacific Grove, California
Control Program/Microprocessor (CP/M) Operating System

International Business Machines Corporation, Boca Raton, Florida
IBM PC Personal Computer
IBM PC/AT Personal Computer

Microsoft Corporation, Bellvue, Washington
Microsoft Disk Operating System (MS-DOS)

RR Software Incorporated, Madison, Wisconson
JANUS/Ada Programming Language

United States Department of Defense
Ada Programming Language

Zenith Data Systems Corporation, St. Josephs, Michigan
Z-248 Personal Computer

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 1

B. PROJECT DESCRIPTION 1

1. Target Hardware 1

2. Network Topology 2

3. Network Media 2

4. Software 2

a. Operating System 2

b. Programming Language 2

5. Proposed Capabilities 2

a. File Transfer 2

b. Distributed Processing 2

c. Control of Multiple Slave Microcomputers 3

d. Remote Login 3

e. Error Handling 3

C. STRUCTURE OF THE THESIS 3

II. HARDWARE 5

A. THE IBM PC/AT PERSONAL COMPUTER 5

1. The Central Processor Unit 5

2. Interrupts 7

3. Communications Ports 8

4. Programmable Interrupt Controller (PIC) 10

III. THE OPERATING SYSTEM 12

A. BACKGROUND 12

B. CHILD PROCESSES 14

1. Program Segment Prefix 14

a. Interrupts to be Restored on Program Termination 15

b. The Environment Segment Address 15

c. File Handle Table 16

d. Redirection 17

C. PROGRAM TERMINATION 19

IV. THE PROGRAMMING LANGUAGE 20

A. JANUS ADA 21

1. Memory Size Limitations of Compiled Code 21

2. Failure of the Child Process Call 23

3. Need for a Replacement Language 25

B. TURBO PASCAL 25

1. Information Hiding 26

2. Support for Child Processes 26

3. Data Abstraction 26

4. Unit Initialization 27

5. Unit Exit Procedures 27

6. Absolute Variables 27

7. File Input and Output 27

8. Port Read/Write 28

9. Interrupt Service Routines 28

10. Exception Handling 28

11. High Level Software Interrupt Procedure..... 28
12. ROM BIOS and Hardware Interrupt Procedures 29
13. Support for a Larger Memory Model 29
C. IMPLEMENTATION 29
V. THE IMPLEMENTATION 30
A. THE HARDWARE CONFIGURATION 30
B. SOFTWARE CONFIGURATION 31
1. The Operating System 31
2. The Distributed Processing Program 31
3. ZCOPY File Transfer Program 32
4. Software Maintenance 32
a. Configuration 32
b. Software Modification 32
C. SYSTEMS DESIGN 32
1. The Command Parser 33
2. The Execution of Child Processes 33
a. Internal Commands 34
b. External (Executable) Commands 35
3. Redirection 35
4. File and Command Transfer via Xmodem 36
5. Serial Communications 37
6. Man Machine Interface 37
D. DESIGN CONSIDERATIONS 38
1. Assembly Language 38
2. ROM BIOS Software Interrupts 38
3. Memory Management 38

4. Synchronization 39
5. Modular Programming 39
6. Preservation of Interrupt Vectors on Program Termination 40
E. SYSTEM EXECUTION 40
1. Initialization 40
2. Slave Operation 41
3. Master Operation 41
a. Terminal Operations 42
b. Port Initialization 43
c. Remote Login to Slave 43
d. Remote Program Execution 43
e. Flow Control 44
f. Remote Reset 44
g. File Transfer 44
F. THE MODULES 45
1. Distrib 45
2. DataCom 45
3. Director 45
4. ErrorCod 46
5. General 46
6. MiscPack 46
7. Parser 46
8. Redirect 47
9. Spawn 47
10. Support 47

11. Wndow 47

12. Xmodm 48

VI. CONCLUSIONS 49

APPENDIX A OPERATOR'S MANUAL 51

APPENDIX B INSTALLATION/PROGRAMMING AIDS 58

APPENDIX C XMODEM PROTOCOL 64

APPENDIX D MAINTENANCE MANUAL FOR DISTRIB PROGRAM 69

APPENDIX E MAINTENANCE MANUAL FOR UNIT DATACOM 81

APPENDIX F MAINTENANCE MANUAL FOR UNIT DIRECTOR 93

APPENDIX G MAINTENANCE MANUAL FOR UNIT ERRORCOD 96

APPENDIX H MAINTENANCE MANUAL FOR UNIT GENERAL 97

APPENDIX I MAINTENANCE MANUAL FOR UNIT MISCPACK 100

APPENDIX J MAINTENANCE MANUAL FOR UNIT PARSER 101

APPENDIX K MAINTENANCE MANUAL FOR UNIT REDIRECT 106

APPENDIX L MAINTENANCE MANUAL FOR UNIT SPAWN 111

APPENDIX M MAINTENANCE MANUAL FOR UNIT SUPPORT 114

APPENDIX N MAINTENANCE MANUAL FOR UNIT WNDOW 120

APPENDIX O MAINTENANCE MANUAL FOR UNIT XMODM 126

APPENDIX P SOURCE LISTING FOR UNIT DATACOM 136

APPENDIX Q SOURCE LISTING FOR UNIT DIRECTOR 156

APPENDIX R SOURCE LISTING FOR UNIT ERRORCOD 162

APPENDIX S SOURCE LISTING FOR UNIT GENERAL 166

APPENDIX T SOURCE LISTING FOR UNIT MISCPACK 171

APPENDIX U SOURCE LISTING FOR UNIT PARSER 174

APPENDIX V SOURCE LISTING FOR UNIT REDIRECT 186

APPENDIX W SOURCE LISTING FOR UNIT SPAWN 196

APPENDIX X SOURCE LISTING FOR UNIT SUPPORT 203

APPENDIX Y SOURCE LISTING FOR UNIT WNDOW 226

APPENDIX Z SOURCE LISTING FOR UNIT XMODM 243

APPENDIX AA SOURCE LISTING FOR PROGRAM DISTRIB 270

APPENDIX AB CONFIGURATION FILE STRUCTURE 301

APPENDIX AC DOCUMENTATION FOR ZCOPY PROGRAM 302

LIST OF REFERENCES 304

INITIAL DISTRIBUTION LIST 306

ACKNOWLEDGEMENTS

Special mention is due to the following individuals who provided solutions to some of the technical problems encountered in the implementation of this thesis, as well as their kind permission to reprint their work as program excerpts used in the thesis.

Diplomate Physics Christian Boettger
Institut fuer Metallphysik und Nuklear Festkoerperphysik der
Technichen Universitaet Braunschweig
Bundesrepublik Deutschland (West Germany) FRG

Reino R. A. de Boer
Erasmus Universiteir Rotterdam
The Netherlands

Naoto Kimura
California State University, Northridge (CSUN)

Alexander Verbraeck
Delft University of Technology
Department of Information Systems
The Netherlands

My greatest thanks are due, of course, to my wife Michelle for her loving support during this entire process.

I. INTRODUCTION

A. BACKGROUND

Many designs for local area networks are currently available on the commercial market, however, all are designed to provide for sharing of high performance centralized assets such as file servers or relatively scarce resources such as specialized printers; or for the movement of data and files. None are known to provide a distributed processing capability by using the inherent capabilities of the attached microcomputers or processors under the control of a central master computer.

The purpose of this thesis is to demonstrate such a capability in a laboratory environment, utilizing a network of slave or server microcomputers capable of running separate applications programs under the control of a central or master microcomputer.

B. PROJECT DESCRIPTION

1. Target Hardware

The proposed demonstration network consists of a single master IBM PC compatible microcomputer connected to two IBM PC or IBM PC/AT compatible slave microcomputers under the operational control of the central master.

2. Network Topology

The proposed demonstration network is a small star network, with the master microcomputer as the central node.

3. Network Media

The proposed networking media shall be the standard RS-232C serial communications port provided with each microcomputer. The central microcomputer is augmented with a second RS-232C serial port to allow independent communications with both slaves.

4. Software

a. Operating System

The operating system selected for the microcomputers shall be Microsoft MS-DOS, version 3.0 or later, as supplied with each microcomputer.

b. Programming Language

All applications software for the microcomputer control programs was originally intended to be written in RR Software Inc. Janus/ADA. The actual implementation is in Borland Turbo Pascal, version 4.0.

5. Proposed Capabilities

a. File Transfer

The master microcomputer shall be able to initiate program and data file transfers to and from any of the connected slave microcomputers.

b. Distributed Processing

The master microcomputer shall be able to command the execution of selected programs resident on any slave microcomputer,

receive an acknowledgment of the command from the slave, and receive the text output of the selected program after execution.

c. Control of Multiple Slave Microcomputers

The master microcomputer shall be able to control more than one slave microcomputer.

d. Remote Login

The master microcomputer shall be able to remotely log in to any slave microcomputer and operate it remotely over the communications network.

e. Error Handling

The master and slave microcomputers shall attempt to restore communications to resume control in the event of a fault.

C. STRUCTURE OF THE THESIS

Since standardized microcomputers and operating systems were selected, the majority of this thesis consists of the programming effort to create the network control programs, and the source code for those programs. What follows will describe the design considerations predicated by the choice of hardware, operating system and programming languages; a description of the significant problems encountered; and instructions for duplicating the network along with program operation and maintenance.

Chapter II will describe the programmer's model of the hardware utilized in the microcomputers and interrupt driven serial communication considerations. Chapter III will discuss the essential features of the operating system as they contributed to the thesis.

Chapter IV will describe the salient features of the two programming languages considered, and the reasons for selecting a replacement for Janus/ADA. Chapter V will discuss the implementation from a systems viewpoint with a brief description of each software module. Chapter VI summarizes the conclusions reached from this thesis.

The appendices provide detailed descriptions of the program source code, the source listings, an operator's manual, a guide to program maintenance, and the bibliography.

II. HARDWARE

A. THE IBM PC/AT PERSONAL COMPUTER

The IBM PC/AT personal computer and its close compatibles, such as the Zenith Z-248 adopted as the standard Navy desktop personal computer, were selected as the target hardware for both program development and application. These computers are general purpose, and typically have at least 640K of random access memory for operating system and program execution, one or more floppy disk drives handling 5-1/4 inch diskettes with 360K bytes of storage each, a hard disk drive holding from ten to twenty megabytes of storage, and a monochrome or color monitor displaying 80 characters by 24 lines of text. One RS-232C serial interface is standard, and a second is optional. The computer also comes with a parallel printer port. The following hardware features are of interest to aid in understanding the software developed: (Norton, 1985, pp. 19 - 65)

1. The Central Processor Unit

The programming model of Table 2.1 is common to the Intel 8088, 8086 and 80x86 series of microprocessors used in the IBM PC/AT compatibles. This information is not provided to support assembly language programming (there is very little in this implementation), but for interface considerations to control, read from, write to, or obtain the status of the IBM PC hardware in support of the distributed processing network. The usage of specific registers for software

interrupts is defined by calling conventions similar to the formal parameter declarations for procedures and functions in higher level languages such as ADA.

TABLE 2.1

MICROPROCESSOR REGISTERS

Register	Type	Function
Scratch Pad Registers:		Arithmetic and data transfer
AX	Accumulator	Arithmetic operations
BX	Base	Table pointer
CX	Counter	Repetition loop
DX	General	General purpose

The above registers may also be addressed as eight bit pairs, i.e., register AX may also be utilized as AL and AH for the low and high order bits.

Segment Registers:		Separate code, data, stack and an extra segment
CS	Code Segment	Locates the code segment in memory
DS	Data Segment	Locates the data segment in memory
SS	Stack Segment	Locates the stack segment in memory
ES	Extra Segment	Intersegment transfers
Index Registers:		Relative offset from a segment register
IP	Instruction Pointer	Points to next instruction to be executed
SP	Stack Pointer	Points to next available location on stack
BP	Base Pointer	Offset into the stack segment
DI	Destination Index	String data transfers
SI	Source Index	String data transfers
Control Functions:		
Flags	Flag Register	Used to record processor status information

2. Interrupts

Interrupts serve two functions in the IBM PC: hardware interrupts allow a peripheral to request servicing from the CPU, and software interrupts allow the operating system or applications software to obtain services from the hardware. Software interrupts are generated by a machine instruction. In either case, a software or firmware interrupt service routine must be called to process the request. The originator of the interrupt does not need to know the address of the routine that accomplishes the service, since the IBM PC incorporates a powerful feature designed to minimize limitations in the inherent design. A level of indirection is designed into the interrupt architecture of the microcomputer that facilitates redefining the interrupt service routines without rewiring the hardware or changing firmware. This is accomplished through a table of interrupt vectors reserved at the first 1024 bytes of system memory. Each of the 256 vector locations is a four byte pointer initialized to point to a specific function by its location in the table. These functions support hardware and software interrupts generated by the CPU (for fault processing), the hardware (for peripheral service), or the operating system or application program (for higher level services). Control is passed to an interrupt service routine by utilizing the vector at the location assigned to that function to call the service routine. By reassigning these vectors through the operating system, the interrupt service routines normally found in the microcomputer firmware may be substituted by another portion of ROM, the operating system or the application program itself.

As an example, the dynamic assignment of interrupt services was utilized to obtain interrupt driven character receive functions in the distributed processing network. Two hardware interrupt vectors pointing to interrupt service routines for the communications ports are assigned to the interrupt vector table at offset $0B for port two (logical port COM2), and at offset $0C for port one (logical port COM1). The distributed processing program developed for this thesis reassigns the indicated vectors to point to interrupt service routines contained in the thesis program itself. These vectors are restored to their previous values on program termination. (Edwards, 1987, p. 195)

3. Communications Ports

The IBM PC is inherently capable of handling up to seven communications ports, but typically is fitted with only two at standardized hardware addresses: logical ports COM1 and COM2. These are capable of data rates ranging from 110 to 38,400 baud; however the microcomputer ROM Basic Input Output System (BIOS) servicing the ports is only capable of setting speeds up to 9600 baud through service interrupt $14. This service interrupt was also replaced by the application program to set the ports and achieve a finer degree of control over their operation than afforded by the BIOS or the operating system. Table 2.2 is correct for an IBM PC (or Zenith Z-248) fitted with two ports (Edwards, 1987, p. 231):

TABLE 2.2

COMMUNICATIONS PORT ADDRESSES

Register	COM1/COM2 Address	Function
Transmit Holding	$3F8/$2F8	Contains the 8-bit character to be transmitted by the port. This is a write only register.
Receive Buffer	$3F8/$2F8	Contains the byte most recently received by the port. This is a read only register.
Interrupt Enable	$3F9/$2F9	A 4-bit register that enables the serial port to generate interrupts to the computer when any of the following events occurs. Bit 0: Interrupt when data are available to be received. Bit 1: Interrupt when the transmit holding register is empty. Bit 2: Interrupt when the line status register changes state. Bit 3: Interrupt when the modem status register changes state.
Line Status	$3FD/$2FD	Provides information about the status of data transfer. Bit 0: Data ready to be received. Bit 1: Overrun error Bit 2: Parity error Bit 3: Framing error Bit 4: Break detected on the line Bit 5: Transmit holding register is empty Bit 6: Transmit shift register is empty Bit 7: Always zero
Modem Status	$3FE/$2FE	Contains the status of the modem signals Bit 0: Delta clear to send Bit 1: Delta data set ready Bit 2: Trailing edge ring indicator Bit 3: Delta line signal detect Bit 4: Data set ready

		Bit 5: Data set ready Bit 6: Ring indicator Bit 7: Receive line signal detect
Line control	$3FB/$2FB	Used to configure the data communications parameters. Bits 0 - 1: Word length (bits): 0 = 5 1 = 6 2 = 7 3 = 8 Bit 2: Stop bits: 0 = 1 1 = 2 Bit 3: Enable parity Bit 4: Select even parity Bit 5: Mark/space parity select Bit 6: Generate BREAK signal Bit 7: Divisor latch access
Modem control	$3FC/$2FC	Allows access to the signals used to communicate with a modem Bit 0: Data terminal ready Bit 1: Request to send Bit 2: Out1 Bit 3: Out2. Must be set to enable UART interrupts Bit 4: Loopback

4. Programmable Interrupt Controller (PIC)

Another programming requirement involved enabling the IBM PC hardware to recognize receive character interrupts generated by the two UARTs. All hardware interrupts are prioritized for the CPU by a device called the Programmable Interrupt Controller. The Intel 8259 Programmable Interrupt Controller is capable of prioritizing up to eight interrupts, identified as IR00 through IR07, with IR00 being assigned the highest (preemptive) priority. The programming requirements are to set the appropriate mask bits in the Interrupt Mask Register of the PIC, and to send an End Of Interrupt command to the device following completion of the interrupt service routine supplied

by the thesis program. Communications port one is assigned interrupt vector IR03 (bit 3), and communications port two has IR04 (bit 4). The 8259 can be instructed to recognize or ignore interrupts from a peripheral by clearing or setting the appropriate bit in the Interrupt Mask Register located at I/O port $21, and this feature was utilized to disable ports when not in use. End Of Interrupt commands are sent to I/O port $20. This relationship is summarized below (Greenberg, 1987, pp. 46-50):

TABLE 2.3

PROGRAMMABLE INTERRUPT CONTROLLER ADDRESSES

Register	Address	Function
Interrupt Mask	$21	Contains the mask for the currently enabled interrupts (read/write) Bit 3: IR03 - Com port 2 mask. Clear to enable the port interrupts Bit 4: IR04 - Com port 1 mask. Clear to enable the port interrupts
In Service	$20	Write to the same bit as enabled in the Interrupt Mask register to clear the interrupt.

III. THE OPERATING SYSTEM

Microsoft MS-DOS version 3.21 was provided with the microcomputers used in this thesis, and provides the traditional functions expected in an operating system: high level interface for applications programs, file services, memory management, and input/output services (MS-DOS Reference Guide, 1986, pp. 2.3 - 2.9). The use of a standard operating system was desirable, as it allowed a piece of "trusted" software to be utilized for most of the distributed processing functions while providing a familiar environment for the operator. Certain extensions to the operating system were constructed in software, to facilitate the execution of programs on the microcomputers and to complement the extensions in hardware services discussed earlier. These are discussed below.

A. BACKGROUND

MS-DOS interfaces directly with the hardware implementation dependent portion of the IBM PC compatible microcomputer, the ROM Basic Input Output System (BIOS). Recall that this BIOS provides a logical interface and some low level services for the underlying hardware, including the disk drives, serial communications ports, keyboard and video display. The ROM BIOS also accomplishes the initialization of the IBM PC on power up. The ROM BIOS services remain available to the programmer through interrupt service calls. (Norton, 1985, pp. 44 - 45)

The portion of MS-DOS that interfaces with the ROM BIOS is contained in a file called IO.SYS, located on the media supplied with the operating system. This file contains extensions and in some cases replacements to the ROM BIOS services supplied with the computer such as device drivers for mouse input devices or specialized video displays not available when the design for IBM PC compatibles was standardized. On initialization, IO.SYS substitutes the replacement interrupt service routines for the existing ROM BIOS services by simply changing the interrupt table vectors to point to the new routines in memory. This facility allows the manufacturer to tailor a standard operating system to various hardware manufacturer's microcomputers. A caution on the means to change these interrupt vectors is noted below. (MS-DOS Reference Guide, 1986, pp. 2.5 - 2.6)

The next file loaded is MSDOS.SYS, which provides hardware independent services for the operating system, i.e., high level interface for file services, memory management, and input/output services. This portion includes the handler for a class of service requests, called DOS function requests, utilized in the distributed processing program to load and execute programs external to the operating system and input/output redirection to implement the capabilities cited in Chapter I. (MS-DOS Reference Guide, 1986, pp. 2.4 - 2.5)

The last portion of the operating system loaded is COMMAND.COM, which builds on the previous layers to provide the familiar command line interpreter and MS-DOS resident commands such as COPY and DIR. (MS-DOS Reference Guide, 1986, pp. 2.7 - 2.9)

The use of function calls to change the interrupt vector table providing ROM BIOS, IO.SYS and MS-DOS interrupt services is strongly encouraged by Microsoft to prevent accidental or malicious corruption of data structures within the operating system and the vector interrupt table. It is also intended to allow backward compatibility for future releases of the operating system that may include multitasking. (MS-DOS Reference Guide, 1986, p. 6.3)

B. CHILD PROCESSES

The ability of the operating system to spawn a local process and regain control after execution is an essential element of the distributed processing network. MS-DOS Function Request 4BH is utilized to load another program into memory and begin execution. Programs executed from the Command.Com command line prompt are executed as child processes of the operating system in exactly the same way. This function provides for the execution of programs and for the remote login capability required by the network. Several details of the MS-DOS operating system capability were of interest in this thesis. (MS-DOS Reference Guide, 1986, pp. 3.1 - 3.9)

1. Program Segment Prefix

When a child process is created, the MS-DOS operating system finds the lowest available segment address to use as the start of program memory for the spawned process, and builds a 256 byte control block called the Program Segment Prefix (PSP) at offset zero within that segment. The executable program immediately follows. While Microsoft does not officially document the use of certain fields within

the PSP, sufficient information was collected from the MS-DOS Reference Guide and other sources to manipulate the environment created for the child process to accomplish the goals of the distributed processing program.

a. Interrupts to be Restored on Program Termination

The interrupt vector table pointers for three essential interrupts are placed in fields of the PSP of the spawned process prior to execution. These are restored on program termination to insure that the interrupt vector table is not corrupted should the child process replace the vectors for its own use and then terminate abnormally. These are: The Terminate Handler Address containing the address of the operating system routine that accomplishes program termination; the Control-C (also known as Control-Break) Address containing the address of the operating system routine that handles operator induced program termination; and the Fatal Error Handler Address used to process errors that result in fatal program halts. (MS-DOS Reference Guide, 1986, pp. 3.5 - 3.9)

b. The Environment Segment Address

The PSP contains a field that holds the segment address of the system environment. This environment is a series of ASCII strings that may be used by programs to determine permissible operations or values. These strings take the form variable = value, and are terminated in a zero (0) character. An example is the "PATH =" environment variable used to set the search paths used by the command processor Command.Com to locate an external command. The process' current environment is made available by following this segment pointer

and searching the strings found at that address until a string with a second terminating zero character is found. This facility is used by the thesis program to locate a copy of the Command.Com on disk to run batch programs (Edwards, 1987, pp. 286 - 288). Each child process inherits a copy of the environment pointed to by the segment address of its parent. This means that the child process may manipulate its own environment without disturbing that of its parent. It also means that the parent may manipulate its own environment prior to spawning a child process in order to communicate with the child or to restrict certain environmental parameters from the child, although this communications means is not reversible. (MS-DOS Reference Guide, 1986, pp. 3.6 - 3.7)

c. File Handle Table

When the PSP is constructed, the operating system places a copy of all open file handles in a data structure of the type FILEHANDLE = ARRAY [1..20] OF BYTE in the PSP (Greco, 1987, p. 25). Each word in the table indexes another data structure internal to the operating system that contains information needed to locate the file on the disk system(s). This inheritance has the effect of passing all the open files of the parent to the child. A file handle is a Unix style 16 bit word that is used to identify a file or a device known to the operating system, and replaces the use of CP/M compatible File Control Blocks for file references by the operating system (Simrin, 1988, p. 204). File handles allow the use of pathnames to open or create a file. Once opened, the file handle is returned to the calling program as the reference to the file. The first five files are opened by the

operating system and have special meaning: (MS-DOS Reference Guide, 1986, p. 5.9)

TABLE 3-1

MS-DOS RESERVED FILE HANDLES

File Handle	Mnemonic	Purpose	Function
0	StdIn	Standard Input	Input can be redirected
1	StdOut	Standard Output	Output can be redirected
2	StdErr	Standard Error	Output cannot be redirected
3	StdAux	COM1	I/O cannot be redirected
4	StdPrn	Printer	I/O cannot be redirected

d. Redirection

Redirection refers to the ability of the input or output character stream associated with one of the reserved files above to be rerouted to or from a different file. An example of this function is the use of redirection characters on the command line (<, >, >>, or |), when program output is redirected to a file or pipe, as in the command line entry: PROGRAM > FILE. When the operating system opens the Standard Error file, it is directed to the same device as the Standard Output file, the display console (logical device driver name CON), and cannot be redirected on the command line as indicated in the table. This limitation would prevent vital error information from being redirected from the slave microcomputer display to the master microcomputer display. (MS-DOS Reference Guide, 1986, p. 3.8)

While such redirection cannot be performed from the program command line, MS-DOS provides function calls that overcome this limitation. These are MS-DOS function calls 45H, Duplicate a File Handle (DUP), and 46H, Force a Duplicate of a Handle (FORCDUP). DUP

creates a new file handle that references the same file at the same position as an existing file handle. It does so by referencing the same internal data structure for the file in the operating system for both files. FORCDUP takes as input two file handles, but forces the first file handle to refer to a file referenced by a second handle. The file referenced originally by the first handle is closed (Simrin, 1988, pp. 450 - 452). To accomplish redirection of the Standard Error character stream and overcome the limitation of the operating system cited in III.A.c above, the parent process may use the following procedure (Greco, 1987, p. 26):

Open the file that Standard Error will be redirected to for writing.

Save a pointer to Standard Error using DUP.

Force the Standard Error handle to point to the newly opened file using FORCDUP. This closes Standard Error.

Close the handle created in (1) since it is no longer needed.

The child program may now be spawned, and has no knowledge of the redirection. Upon termination of the child, the parent reverses the above process:

Force the Standard Error handle to point back to Standard Error by using FORCDUP and the saved pointer.

This redirection method is used for both Standard Error and Standard Output to interleave the two output streams into the same file. A more direct method is to directly manipulate the file handles in the File Handle Table of the Program Segment Prefix, however, this

violates the strictures mentioned in the beginning of this chapter and could corrupt the data structures contained in the operating system if improperly done. The use of documented function calls allows the operating system to protect itself and to provide error handling.

C. PROGRAM TERMINATION

Upon termination of the spawned program, the operating system accomplishes the following (MS-DOS Reference Guide, 1986, p. 4.241). First, the three interrupt vectors described above are restored to the interrupt vector table from values stored in the terminated process' PSP. Next, control is given to the Terminate Handler address to return control to the invoking process. Finally, all open files are closed. Recall that the calling program retains a copy of all open files in its own PSP. The effect of closing all the files of the child is to flush file buffers held internal to the disk operating system and update the disk directories (Defenbaugh, 1986, p. 22). The operating system then terminates any redirection.

IV. THE PROGRAMMING LANGUAGE

Implementation of this thesis was originally attempted in a subset of the Department of Defense programming language mandated for mission critical computer resources, Ada. Ada was chosen to explore the language in this environment and to apply the language features that localize the major design decisions into individual program modules (decomposition), promote information hiding through separate compilation, and support data abstraction. Concurrency might have allowed the separation of the communications and control requirements into separate tasks, but was not supported in the subset. (MacLennan, 1987, pp. 261 - 263)

The subset of the Ada language chosen for this project was RR Software Inc. JANUS/Ada. This subset of the approved language had several limitations in addition to the lack of concurrent programming (task) facilities, but was available and could be utilized on the same microcomputer for program development and implementation. It had been used successfully in a similar environment for local area networking (Works, 1986), (Hartman and Yasinsac, 1986), and includes a very capable assembler for constructing machine language packages. It turned out that this particular implementation was unsuitable to the proposed capabilities of the distributed processing network for the reasons cited below.

A. JANUS/Ada

1. Memory Size Limitations of Compiled Code

The initial work for this thesis was to construct a command line parser to recognize commands in MS-DOS syntax for execution on the slave microcomputer. This was first implemented in assembly language following the program of an established command intercept processor (Mefford, 1986, pp. 313 - 334). This program successfully parsed the elements of a command line and reported these components, thereby demonstrating the potential to execute the command remotely. The code files of table 4.1 resulted. Files ending in a "jrl" suffix are compiler relocatable object files and files ending in a "com" suffix are the linked result suitable for execution.

TABLE 4.1

ASSEMBLY LANGUAGE PARSER

Program Name	Language	File Size (bytes)
find_com.jrl	assembly	791
parsemai.jrl	Ada package	148
parsemai.com	compiled	4480

The parser was then recoded as an Ada package to obtain the flexibility of the higher order language and to develop the assembly language to Ada package interfaces. JANUS/Ada allows assembly language procedures to call Ada procedures and functions, and to reference Ada data structures. The implementation of the parser as an Ada package allowed rapid modification to the parser to adjust the command syntax, as well as for interface to the other Ada packages to be developed for the system. When compiled, however, the following resulted:

TABLE 4.2

Ada LANGUAGE PARSER

Program Name	Language	File Size (bytes)
Int_21.jrl	assembly	948
cmdlyne.jrl	Ada	13656
main.jrl	Ada	505
main.com	compiled	42423

The cost of coding in this implementation of JANUS/Ada is evident above. The JANUS/Ada compiler emits about a tenfold increase in code size to accomplish the same effort as the assembly language version. The COM file is also much larger, due to the incorporation of library routines from the Jlib86 support package to handle string manipulation and other high level language constructs. With a code size limitation of 64K bytes, results similar to the above would rapidly exhaust the space available in the small memory model as packages were added. This model is limited to 64 Kbytes of code and a separate 64 Kbytes of data (JANUS/Ada Package User Manuals, 1983, p. Z - 4), and is characteristic of COM files running under MS-DOS. The options were either to code major portions of the thesis in assembly language as had been done by Works, Hartman and Yasinsac, linked together by Ada packages as a main program, or to find a way to expand the code module. The latter was desirable due to the original intent to utilize a higher level language for the distributed processing network. Before this could be pursued, however, a more serious problem developed.

2. Failure of the Child Process Call

As described in Chapter III, MS-DOS commands or programs not implemented internally by the operating system are called transient commands, and must be run by loading the program into memory from disk and executing it as a child process. As the next step in the above implementation, a call was constructed in an assembly language package body to the MS-DOS function 4BH, EXEC program (MS-DOS Reference Guide, 1986, pp. 4.237 - 4.239). This was done to overcome a limitation of the JANUS/Ada supplied procedure, Prog_Call. The supplied procedure recognizes only program names without path specifications, and does not allow for a command tail after the program name. The procedure also terminates both the child process and its parent if the child process terminates abnormally. This would not allow for a robust distributed processing system, capable of recovering from a faulty child program and continuing to operate in the network (JANUS/Ada Package User Manuals, 1983, p. 15 - 3).

When this approach was implemented, however, all child processes would execute normally when called from the MS-DOS function, as expected. The system would lock up upon return of control to the parent process, usually with a fatal error message such as INTERNAL STACK OVERFLOW. This suggested that something was being corrupted in the MS-DOS operating system upon termination of the child program.

An investigation of a disassembly listing of the compiled program revealed that the JANUS/Ada runtime library was writing initialization data into reserved areas in the Program Segment Prefix of the parent program. These areas are undocumented by Microsoft in

its official literature, but have been identified by other authors. Table 4.3 shows these locations: (Simrin, 1987, p. 211 - 212)

TABLE 4.3

JANUS/Ada INITIALIZATION AREAS

Location	Contents
PSP:0016	PSP of parent process
PSP:001C	Standard Printer file handle (filehandle[4])
PSP:001E	filehandle[6]
PSP:0020	filehandle[8]
PSP:0022	filehandle[10]
PSP:0024	filehandle[12]
PSP:0026	filehandle[14]

Since the filehandles are indices to data structures internal to the operating system holding information about specific open files, the consequences of these actions are that the compiled program unintentionally creates open filehandles after the Standard Printer handle assigned by MS-DOS, or overwrite the filehandles for files already opened by the parent program. Recall that MS-DOS opens the first five handles, and the application program opens filehandles after that up to the FILES = <number> set in the environment. When the JANUS/Ada program overwrites these handles, the indices represented by them now point to other potentially unrelated areas of the operating system for files referenced by the file handles. These other areas may then be corrupted when the operating system attempts to close the child process' files using invalid file handles. These data structures are common in the operating system to both parent and child. This may explain why the JANUS/Ada built in file operations and functions would no longer work after a single assembly language call to operating

system function calls, as observed by Works (Works, 1986, p. 24). Works wrote all file handling procedures for his program in assembly language to overcome this fault. (Works, 1986, p. 33)

The effect of corrupted data areas in the operating system is to compromise the internal state of MS-DOS when the child process terminates.

3. Need for a Replacement Language

At this point, a decision was made to implement the thesis in a language that would support child processes and provide a larger memory model.

B. TURBO PASCAL

While performing the initial work for this thesis, Borland Corporation Turbo Pascal version 3.0 was being examined for the possible use of a construct similar to its operating system calls. The language utilizes a very general procedure to call MS-DOS functions and software interrupts with a data structure standing in for the contents of the microprocessor registers discussed in Chapter II. With such a procedure constructed for the JANUS/Ada language as a supporting package, the large number of assembly language procedures and functions that Works, Hartman and Yasinsac required could be abstracted out to a single general purpose procedure, tailored for each instance by the register contents.

When the difficulty encountered with the failure of child processes in JANUS/Ada, a rapid prototyping effort was used in Turbo Pascal to check the author's understanding of the requirements for the EXEC call

in another language to detect possible errors in implementation. The EXEC function worked satisfactorily in Turbo Pascal, using either the MS-DOS call construct or the compiler's built in procedure. Since the Ada implementation appeared to be infeasible, the program was implemented in Turbo Pascal. It turned out that version 4.0 of that language has features that capture the essence of the original programming objectives. Some particular features follow:

1. **Information Hiding**

Borland's Turbo Pascal version 4.0 implements the Unit as originally developed for UCSD Pascal (Duntemann, 1987, p. 11). This programming construct allows modular programming very similar to Ada, however separate compilation cannot be achieved with just the module interface declaration, as it can in Ada. Variables and procedures implemented in the UNIT body are not visible by outside modules, as in the Ada package.

2. **Support for Child Processes**

Turbo Pascal provides a robust implementation of the MS-DOS Function 4BH, called EXEC. This is a high level procedure that takes Pascal strings for the program path specification and the command tail arguments as parameters. The procedure utilizes the Turbo Pascal global variable DOSError to report operating system error messages for program handling.

3. **Data Abstraction**

Turbo Pascal supports data abstraction in much the same way as Ada, but does not implement a Private declaration.

4. Unit Initialization

The Turbo Pascal Unit provides an initialization section for Units, which can be used to perform unit configuration and to save state information prior to program execution. This is helpful for saving interrupt vector table contents for restoration on program exit.

5. Unit Exit Procedures

Turbo Pascal provides an important feature by allowing the programmer to declare an exit procedure that will be run upon program termination. This procedure will execute for normal or abnormal termination, and can be constructed to provide error handlers. The primary use in this thesis was to insure that interrupt vectors were properly restored on program termination.

6. Absolute Variables

Turbo Pascal supports manipulation of hardware memory locations by allowing the programmer to specify the actual location in memory of a data structure. This is accomplished by the ABSOLUTE reserved word in a VAR declaration, and was used to declare a pointer to reference the video memory for windowing operations (Edwards, 1987, p. 30).

7. File Input and Output

Turbo Pascal provides the capability to read or write to untyped files in addition to Wirth's Read and Write procedures. This allowed the file transfer protocol to treat a file as a stream of bytes.

8. Port Read/Write

Turbo Pascal provides Port and Portw procedures to read or write byte and word sized variables to the IBM PC ports. This capability was used in the serial communications port module.

9. Interrupt Service Routines

The Turbo Pascal compiler has a special reserved word, INTERRUPT, that allows the programmer to define procedures as interrupt service routines. The compiler handles all register preservation and stack operations across the call.

10. Exception Handling

Turbo Pascal does not implement the Ada exception handler, however, the combination of the DOSError variable and the ability to relax I/O, range and type checking within a local scope allows the programmer to place the exception handling mechanism in the control flow with standard structured programming techniques. An EXIT procedure with a scope identifier would have been useful to escape a procedure, however, the current approach enforces structured programming.

11. High Level Software Interrupt Procedure

Turbo Pascal provides a predefined procedure, MSDOS, and a data type, registers, that allows a simple and standardized interface to the operating system software interrupt function calls. The registers data type stands in for the processor's built in registers and allows the programmer to treat the MS-DOS functions in the same manner as a procedure. No assembly language programming is involved.

12. ROM BIOS and Hardware Interrupt Procedure

The above procedure, MSDOS, is a special case of the general Turbo Pascal procedure, Intr (Intr, regs), which allows access to any hardware or software interrupt available on the IBM-PC compatible microcomputer. No assembly language programming is involved.

13. Support for a Larger Memory Model

Turbo Pascal compiles programs into EXE files, and greatly expands the potential size of a program. Each unit has an independent code segment, with a maximum size of 64 Kbytes. A single data segment and stack segment is allowed, each with their own 64 Kbyte limitation. The remainder of memory, up to 640 Kbytes, is available on the heap. The stack and heap size may be set by compiler directive to leave room for spawned processes. (Duntemann, 1987, p. 12)

C. IMPLEMENTATION

The distributed processing program was implemented in Turbo Pascal 4.0, as described in the next chapter. This language provided support for all proposed capabilities while eliminating the requirement for extensive assembly language programming.

V. THE IMPLEMENTATION

The distributed processing program in this thesis has its origins in an existing terminal program supporting the Xmodem protocol (Edwards, 1987, pp. 220 - 275). This "brassboard" program served as the foundation for the addition of the command transfer functions that were required by the proposed capabilities of the distributed processing network, and was expanded to provide finer control over multiple serial ports. In addition, command parser and local execution modules were added for the Slave microcomputer to execute resident programs. The operator interface and windowing environment was largely retained intact, and is utilized for the man machine interface.

This approach allowed the referenced program to be modified in discrete steps, and provided a test environment to exercise each portion of the implementation listed below.

A. THE HARDWARE CONFIGURATION

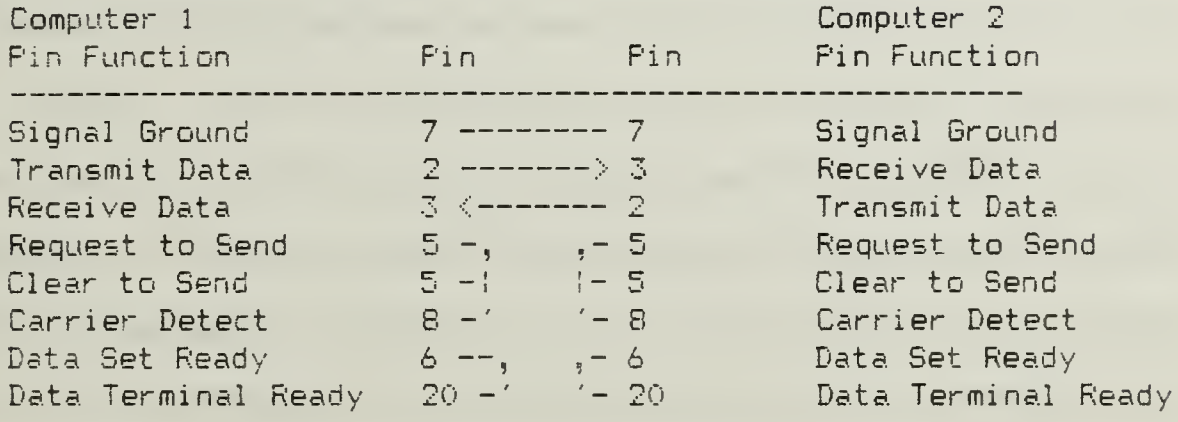
The hardware used to implement the distributed processing network consists of IBM PC/AT compatible microcomputers. Each Slave microcomputer is supplied with a hard disk drive of 10 megabytes or greater capacity, 640 Kbytes of memory and one RS-232C port. The Master microcomputer is configured identically, except it has an additional communications port.

The serial connection between computers are the RS-232C communications ports operating at 9600 baud for IBM PC/AT compatible

machines and 4800 baud for IBM PC/AT compatibles. The microcomputers at each end of a single link must be configured for the same speed. The pin connection for the interconnecting cables is shown at Figure 6.1. For microcomputers with the nine pin AT style connector, a nine pin to RS-232C 25 pin DB-25 cable is recommended, with a NULL modem in between. Hardware handshaking is turned back in this configuration. The program will operate satisfactorily through a modem if the baud rate is lowered. (Flanders, 1989, p. 252)

FIGURE 5.1

SERIAL PORT CONNECTIONS



B. SOFTWARE CONFIGURATION

1. The Operating System

The operating system is supplied with the microcomputers, and is Microsoft MS-DOS, version 3.0 or higher.

2. The Distributed Processing Program

The distributed processing program was written to accommodate the above operating system, and is used on both the Master and Slave microcomputers.

3. ZCOPY File Transfer Program

A high speed, adaptive file transfer program is provided with the distributed processing system software that allows file transfers to be executed at the maximum speed permitted by the serial communications link. The maximum speed is 115 Kbytes/second. The program runs as a child process under the distributed processing system, and includes independent error checking protocols. (Flanders, 1989, p. 282).

4. Software Maintenance

a. Configuration

Configuration is accomplished by a built in function in the program, provided the program was initialized as a Master. This normally suffices to set default configuration options, such as port settings, for automatic loading when the program is run. The settings are saved in a file. If the file is erased, the program initiates its default settings and the operator can then recreate the file.

b. Software Modification

Software modification is accomplished through built in editing, compilation, and run time environment supplied with Turbo Pascal version 4.0. Build and make utilities are supplied with the compiler to allow program modification and rebuild.

C. SYSTEM DESIGN

The problem of designing a distributed processing network was decomposed into the following efforts:

1. The command parser for the remote (slave) microcomputer.

2. The execution of child processes.

3. Redirection of child process output.

4. File and command transfer via Xmodem.

5. Serial communications.

6. The man machine interface.

1. The Command Parser

The command parser decomposes an MS-DOS command directed to the Slave microcomputer for execution into its component disk drive, path, command or executable file name, and command arguments. The latter is commonly called the command tail. Since compatibility with the current MS-DOS command syntax was desired, these commands take the form:

[drive:][\][directory\]..[directory\] command [command tail]

Once parsed, the type of command is determined so that the Slave computer can execute it properly. As an experiment, the Unix commands CAT and LS are mapped into their MS-DOS equivalents to demonstrate a Slave with limited bilingual capabilities.

2. The Execution of Child Processes

Once the command is parsed, the parser must properly determine if the command cited is a command normally executed internally by MS-DOS, an executable COM or EXE file, or refers to a directory operation. Internal MS-DOS commands implemented within the distributed processing program are detected by pattern matching, the remainder are identified by conducting an iterative search across the specified directory (or the current directory if none is cited in the remote command) for an executable file of the appropriate extension, utilizing

the Turbo Pascal built in functions Find_First and Find_Next. If found, the type of file is passed by the parser to the appropriate execution routine. The executable files are those with COM, EXE or BAT extensions. MS-DOS does not require the operator to enter the extension, and will execute the first file encountered with the command name and an executable extension in the following order: COM, EXE and BAT. The parser copies this trait. Implementation of the different command types is summarized below.

a. **Internal Commands**

Internal commands are those that are executed within the MS-DOS command processor, and are available from the familiar A> prompt. These include the directory manipulation commands ChDir, Copy, Del, Dir, MkDir, Ren, RmDir and disk drive login; to which were added a prompt command to obtain the current directory on the Slave microcomputer for display at the remote, and Equip, which provides the Slave configuration (disk drives, memory, etc) accessible to the ROM BIOS interrupt #11 (MS-DOS Version 3.21 User's Guide). ChDir, MkDir, and RmDir along with Prompt are provided within the distributed processing program. Error messages are supplied from the MS-DOS operating system, hence, they are identical to those encountered in local operations. Rather than duplicate the capabilities of the MS-DOS command processor for the remaining commands, MS-DOS is utilized to assist in this effort. A secondary copy of the MS-DOS command processor is located by inspecting the "COMSPEC=<path/name>" string from the local environment area, and is spawned with the appropriate command tail for the desired command. This allows the remote command

to execute as if it were entered from the Slave microcomputer's keyboard, and provides a familiar response. A utility program in the public domain was utilized as a programming template to detect the proper course of action before spawning a child process, depending on the type of command received. (Mefford, 1988, pp. 321 - 336)

b. External (Executable) Commands

External commands are those that require the distributed processing program to load, execute and collect output for display. These are the familiar COM, EXE, and BAT files found in directory listings. These commands are executed by calling the Turbo Pascal EXEC procedure directly from the distributed processing program, with the explicit path specification required by the procedure supplied by the parser in its search for the executable file. The command tail is provided from the parsing operation. Batch files are handled by spawning a secondary copy of the command processor with the batch file name as the command tail, as described for selected internal commands. (Mefford, 1988, p. 327)

3. Redirection

Redirection control is contained in a separate module that contains most of the Turbo Pascal EXEC calls. Prior to spawning an executable file, a variable is checked to determine if the program output is to be redirected to a file managed by the distributed processing program. This file is used to send the program output back to the Master microcomputer over the communications channel by the Xmodem protocol after execution of the program cited in the remote command. The variable is managed by the module initilization routines

and is normally set for redirection, otherwise the program output would appear on the Slave microcomputer screen. If redirection is desired, the distributed processing program redirects its own output to the redirection file, utilizing the MS-DOS Function Calls 45H (DUPlicate handle) and 46H (FORCDUPlicate handle) as described in Chapter III. Since the child process inherits all open files from the parent (in this case the distributed processing program), it proceeds through the execution oblivious to the redirected output. Error reports are also available in the redirected output file, which overcomes a limitation of redirection invoked from the command line with the <, >, >> and | symbols. The appropriate files are then available to forward to the Master microcomputer. (Greco, 1987. p. 25)

4. File and Command Transfer via Xmodem

Since the Xmodem protocol is utilized for both command and data transfer, the highly modularized approach found in (Krantz, 1985, pp. 66 - 89) is implemented to handle synchronization, packet transfer, and file transfer under flow control in a hierarchical manner. The modular approach does require a large number of variables that are global in scope to the different building blocks, however, the concentration of these variables and their associated function and procedure implementations in a Turbo Pascal Unit as private variables preserved information hiding. An additional file transfer program, Zcopy, is available as an operator option on the Master display and allows the use of an adaptive protocol that transfers files at the maximum speed of the communications link, regardless of settings.

5. Serial Communications

All communications between the Master and Slave microcomputers are handled by the microcomputers standard serial communications ports. Communications is at 9600 baud for communications between IBM PC/AT compatibles, and at 4800 for IBM PC compatibles. The interrupt service routines handle receive character streams for hardware ports COM1 and COM2, and are adapted from source listings posted on the info-pascal@vim.brl.mil network (Kimura, 1988) and (de Boer, 1988). Receive characters are queued in a receive buffer for each port. Transmit characters are sent under program control in a polling loop.

6. Man Machine Interface

The program uses the same operator interface for both the Master and Slave configurations. Initialization is accomplished from a configuration file in the local directory or from default constants if the file is absent. When initialized, the program presents a terminal screen for the primary port with communications inhibited. The operator is then able to select options by special key combinations (Alt-keys) to revise the configuration file, initialize communications ports, enable and disable receive interrupts on a port basis, and select the current port for use with file transfers and command transfers to the connected slave. File, command transfers, and the output of the remote Slave computer is available on a monitor window. Status windows are shown for critical parameters.

The Slave microcomputer is operated in an infinite loop to receive and process commands. Local operation may be restored (at the

cost of disabling server functions) by pressing a local key which aborts the Slave program.

D. DESIGN CONSIDERATIONS

1. Assembly Language

Assembly language is used in only two locations in this thesis, for the purposes of code optimization. The first is to move data between the screen buffer and a storage location to open and close windows on the screen as used in the windowing module. The second is to enable and disable CPU interrupts for the interrupt service routines contained in the data communications module. Both instances utilize built in assembly language facilities of the compiler. The remainder of the program is coded in the Turbo Pascal dialect.

2. ROM BIOS Software Interrupts

Calls are made to the ROM BIOS of the IBM PC compatible computers to perform communications port speed initialization (interrupt $14), and to obtain the machine disk drive, memory, and communications port configuration for display ($11).

3. Memory Management

Memory management is handled by the Turbo Pascal compiler in accordance with the $M compiler directive. This was adjusted from that offered by the Turbo version 3.0 to version 4.0 conversion utility, which allocated all memory to the distributed processing program. By reducing the size of the heap, child processes and MS-DOS shells can be run from the program as a parent. The primary consumer of heap memory is for dynamic allocation of memory to save screen

displays for windowing. Current program memory requirements are less than 75 Kbytes, exclusive of the MS-DOS operating system and any Terminate and Stay Resident programs run before the program. The use of Terminate and Stay Resident programs is not recommended due to unpredictable side effects.

4. Synchronization

Synchronization is normally maintained by starting the Slave microcomputer in the command receive mode and then executing in an endless loop. The Master computer operator must initialize the communications ports (if required) and connect to the appropriate port to access the desired Slave. Commands are normally passed to the Slave and responses displayed on the Master, however, if the Master computer is redirected to another task while the Slave is processing the request, the Slave will wait on the Master with its response. This is a functionality of the Xmodem protocol, which is receiver driven. A resynchronization command is available to the Master operator to force the Slave back into the command receive mode if required. The process is currently manual, and depends on operator familiarity with the likely Slave responses. Adequate, although not necessarily automated, status responses are available to the Master operator to determine the Slave state.

5. Modular Programming

The windowing support unit, the Xmodem file and command transfer protocol, and the RS-232C serial communications port and interrupt service routines are contained in separate units. In the case of the Xmodem unit and the data communications unit, the original

terminal program interface is retained although the implementation is considerably different. This was intentionally done to create the potential to provide a different transfer protocol or to use a different network by redesigning the implementation section of the unit, and to demonstrate information hiding. The windowing unit was simply converted to a Turbo Pascal unit (Edwards, 1987, pp. 50 - 98), along with a general support unit (Edwards, 1987, pp. 66 - 73).

6. Preservation of Interrupt Vectors on Program Termination

The manipulation of the vectors in the IBM-PC interrupt vector table provides a powerful means to enhance the capabilities of the machine, whether to incorporate new hardware or to adapt an existing capability in software. The potential is equally high to lose control of the system if the interrupt vectors are not restored when the program ends. This must be handled for normal termination as well as unplanned, or abnormal termination.

E. SYSTEM EXECUTION

1. Initialization

The program contains all functions for operation as either a Master or Slave microcomputer on the distributed processing network. The operating selection is made when the program is run, either by

```
Distrib Master
```

for operation as a master, or by

```
Distrib
```

or

```
Distrib Server
```

for operation as a Slave. The program than searches for its configuration file and uses that to set the default communications port settings, screen colors, etc. If not found, the program utilizes built in defaults.

2. Slave Operation

Slave operation is automatic, with the program initializing its communications port (default is normally COM1), and entering the command processing mode in an infinite loop. This loop may be reset by the remote Master if the Slave is expecting to return a sequence of responses from a completed command, and the Master operator decides to abandon the command after execution. In this case, the Slave is reset over the communications port to the beginning of the command receive loop to prepare for the next command. The program is aborted and control is returned to the operating system if any key on the Slave keyboard is depressed. No warning is sent to the Master, since the Master may be communicating with another Slave and receive buffers are purged to begin a new communications sequence as recommended in the Xmodem protocol. The Master operator can check for a "live" Slave by watching for the received NAK characters, displayed each five seconds over the receive channel, or enter receive mode to display a program response from the Slave.

3. Master Operation

Master operations are menu driven. Upon initialization, the Master displays a status bar showing the current communications port selected at the bottom of the screen and queues the operator to depress the HOME key for a list of commands available. The program otherwise

displays a blank terminal screen although the communications ports are disabled for receive on startup. When the operator depresses the HOME key, a window appears that offers the following command selections with a menu bar that can be positioned to select the desired command. The operator is also reminded that the listed commands may be selected from the terminal screen by depressing the Alt - <key> combination. The commands are:

```
Alt-A  Change drive & path
Alt-B  Send a Break signal
Alt-C  Update Config File
Alt-D  Dialing Directory
Alt-E  Local echo toggle
Alt-F  Change DC params
Alt-G  Show disk directory
Alt-H  Hang up phone
Alt-L  DOS Shell
Alt-M  Activate Master
Alt-P  Port Operations
PgDn,
Alt-R  XMODEM Get a file
Alt-S  Activate Server
PgUp,
Alt-T  XMODEM Put a file
Alt-X  (ESC) Exit emulator
```

A more complete discussion of the different commands is found in Appendix A, the Operator's Manual. The following is a summary of capabilities, as seen from the Master microcomputer.

a. Terminal Operations

The opening screen of the program is adequate to perform teletype terminal communications over the currently selected communications port, once properly initialized. The initialization commands are found in the Activate master subscreens.

b. Port Initialization

The menu selections available allow the operator to override the default communications ports settings and to select a communications port for communications with the remote Slave. An ESC key returns the operator to the terminal screen.

c. Remote Login to Slave

Most operations are accomplished at the Slave by using the remote login function. The command is packetized at the Master and sent to the Slave as a 128 byte Xmodem packet. Upon successful receipt at the Slave (signalled by an ACK character received at the Master), the Master then assumes the Xmodem receive function to await the response from the Slave. The Slave then sends a packet back with a prompt containing its current directory and drive. This prompt is structured to look like the operating system prompt. Once received by the Master, the Slave reverts to command receive mode to await the next command. The Master displays a window to prompt the operator for the next command to send to the Slave, or to quit the command mode. If a command is sent, it is packetized and transmitted as before.

d. Remote Program Execution

Programs are run on the Slave microcomputer in response to commands received from the Master. Once the command is parsed, the program handles some commands internally and runs a program as a child process to accomplish those commands it does not recognize internally. For spawned programs, the program output is captured in a file and then sent back to the Master. The Master waits for the response after

sending the command. Responses may be a series of strings or files, and are displayed on the Master remote login window.

e. Flow Control

Flow control (selection of receiver and sender) is in accordance with the Xmodem protocol, with one exception. An EOT (End of Transmission character is specified in that protocol to signal a complete transmission. In order to accomplish multiple string or file transmission from the Slave to the Master to forward the output of a spawned program, the Master interprets each received EOT character as an end of transaction (string or file) as in the original protocol, but does not end its receive operations until a CAN character is received from the Slave to signal the end of the command and response sequence.

f. Remote Reset

Related to flow control is the ability for the Master microcomputer to reset the flow direction if the Master and Slave microcomputers lose synchronization. This may happen between the command transfer to the Slave and the response from that microcomputer, and is usually exhibited by both microcomputers attempting to send or receive at the same time. The Master operator may break the deadlock by sending a series of CAN characters to the Slave to force it back into the command mode.

g. File Transfer

To send a file, the operator selects the ZCOPY option to the remote microcomputer and the system prompts for a filename. A complete path may be specified. Once selected, the program invokes a copy of the ZCOPY program at the Slave and places it in ZCOPY Server

mode. The Slave then waits for the handshaking protocol from the ZCOPY program at the Master (also spawned), and establishes a link over the serial port at the maximum reliable data rate. Once the transfer is complete, both copies of ZCOPY terminate and control is restored to the distributed processing program at the established data rates. The Slave then reports the ZCOPY program output to the Master.

F. THE MODULES

The following program modules are contained in the distributed processing program.

1. Distrib

Distrib is the main program for both the Master and Slave computers.

2. DataCom

Unit DataCom provides all procedures and functions needed to initialize the computer serial communications ports, enable and disable receive interrupts, provide buffered reception of characters, clear the receive buffer(s), send or receive bytes through the ports, send a BREAK signal over the RS-232 port, and nondestructively read the receive buffer(s). It supports Unit Xmodem and the terminal portion of Distrib. The currently selected communications port is contained in public variable Current_Com.

3. Director

Unit Director is a set of functions and procedures that allow the output MS DOS file directories to a windowed environment. Masking

options and a selector for normal or abbreviated (similar to the MS-DOS /w switch) displays are allowed.

4. ErrorCod

ErrorCod is a array of string constants mapped by the DOS Error Code, Error Class, Recommended Error Action and Error Locus indices found in (Microsoft, 1986, pp. 3-1 - 3.11, 4.254 - 4.255). The unit is used by the units Parser, Spawn and the program Distrib to report errors. A procedure is also provided to retrieve extended error code information available in MS-DOS versions 3.0 and above by DOS function call 59H.

5. General

The General Unit is a collection of general purpose routines that support the Wndow Unit and other modules. (Edwards, 1987, pp. 66 - 73)

6. MiscPack

Unit Miscpack is a collection of data types and utility routines supporting these other units: Xmodm, Parser, Spawn, Redirect, and the main program Distrib. The strong typing features of Turbo Pascal require that instances of data types in different units that must be equated be declared in one place to be compatible at compile time. (Swan, 1986, pp. 14 - 23)

7. Parser

Unit Parser contains a central procedure, Parser_Main, which attempts to parse and execute an MS-DOS style command on the local machine. The remaining procedures and functions support this function.

8. Redirect

Unit Redirect is a set of functions and procedures that allow the output of programs spawned under the Slave computer's copy of the main program Distrib to be redirected to files. Once the program ends, the Slave computer can then forward the output normally displayed on the screen to the Master computer for display.

9. Spawn

This Unit detects commands that should be processed internally by the Distrib program, and executes commands internally or by spawning a child process. Command output and error responses are returned to the caller either as strings suitable for conversion to Xmodm packets, or by reference to files containing the text. This unit also contains the redirection switch as a public variable that dictates whether program output will be redirected to a file or displayed locally on the screen. This switch is normally set to redirect to file.

10. Support

The Support Unit contains most of the constant declarations for the program, along with the initialization procedure some general purpose procedures as found in the original terminal program. (Edwards, 1987, pp. 241 - 272)

11. Wndow

The Wndow Unit provides all window creation, memory allocation, display, menu bar processing, closure and memory deallocation functions for the program Distrib. The unit was changed from an include file to a unit, but not otherwise changed from that

originally developed by the author in (Edwards, 1987, pp. 50-98). The purpose descriptions are from the author.

12. Xmodm

This Unit handles all requests for Xmodem protocol packet and file transmission and reception.

VI. CONCLUSIONS

The program developed and implemented for this thesis successfully demonstrated the capability for unmodified IBM PC/AT compatible microcomputers to operate in a distributed processing network. A small star network consisting of one master microcomputer and two slave microcomputers was installed and operated in a laboratory environment.

The network displayed the capability of transferring program and data files between the master microcomputer and either of the slave microcomputers, and the capability of the master to command the execution of MS-DOS commands and executable programs on the slaves. The network further demonstrated that the output of the commands or programs could be displayed on the master computer. A simple error recovery methodology was also demonstrated.

Implementation of this program was not feasible in RR Software, Inc. JANUS/Ada, due to unexpected problems in the implementation of that subset of the Ada language and that compiler's design. This is not a fault of the Ada programming language. These design deficiencies in the JANUS/Ada were specific to the implementation in an MS-DOS or CP/M environment; and caused fatal operating system faults when a child process was executed from the command parser, as implemented in JANUS/Ada. The amount of code emitted by the compiler also appeared to be relatively large. It should be noted that the compiler available for this thesis was relatively old, version 1.5.2, and as a subset of

the Ada language was not validated. It may be that the current, validated version has corrected these deficiencies.

Borland Corporation. Turbo Pascal proved to be a viable programming environment for this thesis, and provided many of the features desired from the Ada programming language. These include information hiding through modular program and the unit structure, data abstraction, strong typing, and high level procedures for file input and output, access to the microcomputer input/output ports, and a standardized interface to the system software interrupts. Assembly language programming was not required, and was used in two isolated locations to implement replacement interrupt service routines and enhance block data movement.

APPENDIX A

OPERATOR'S MANUAL

A. STARTUP

The distributed processing program is designed to operate on an IBM PC/AT compatible microcomputer such as the Zenith Z-248. Minimum configuration is a 10 Mbyte or larger hard drive, 640 Kbytes of memory, an EGA or VGA monitor, and at least one floppy for program loading. The following files should be resident on the hard disk in the desired directory: DISTRIB.EXE, DISTRIB.CFG, DISTRIB.PHN. A subdirectory should exist off the root directory of the hard disk named SCRATCH for the maintenance of redirected output files generated by the Slave program. The file transfer program ZCOPY.COM should be available in the DISTRIB.EXE directory.

The microcomputers must be connected by a null modem and appropriate cables before the network will operate. Turn on the Slave microcomputer(s) first.

B. Slave Operation

Slave operation is automatic. For convenience, if the microcomputer is to be used largely as a Slave in the distributed processing network, an AUTOEXEC.BAT file may be placed on the boot drive root directory that specifies the complete drive and path specification for the program, with the following program name:

[drive][path]DISTRIB Server

On startup, the program will load, initialize and display a status screen with a monitor window for remote commands and the Slave's responses. Operation of the Slave may be monitored from the display screen. The program is aborted and control is returned to the operating system if any key on the Slave keyboard is depressed. No warning is sent to the Master.

C. Master Operation

Master operations are menu driven. For convenience, if the microcomputer is to be used largely as a Slave in the distributed processing network, an AUTOEXEC.BAT file may be placed on the boot drive root directory that specifies the complete drive and path specification for the program, with the following program name:

[drive][path]DISTRIB Master

On startup, the program will load, initialize and display a status bar at the bottom. This bar shows the current communications port selected at the bottom of the screen and queues the operator to depress the HOME key for a list of commands available. The program otherwise displays a blank terminal screen although the communications ports are disabled for receive on startup. When the operator depresses the HOME key, a window appears that offers the following command selections with a menu bar that can be positioned to select the desired command. The operator is also reminded that the listed commands may be selected from the terminal screen by depressing the Alt - <key> combination. The commands are:

```
Alt-A  Change drive & path
Alt-B  Send a Break signal
Alt-C  Update Config File
Alt-D  Dialing Directory
Alt-E  Local echo toggle
Alt-F  Change DC params
Alt-G  Show disk directory
Alt-H  Hang up phone
Alt-L  DOS Shell
Alt-M  Activate Master
Alt-P  Port Operations
PgDn,
Alt-R  XMODEM Get a file
Alt-S  Activate Server
PgUp,
Alt-T  XMODEM Put a file
Alt-X  (ESC) Exit emulator
```

These commands are discussed individually in the following sections. What follows is a general sequence of commands or selections to accomplish processing on the Slave microcomputer.

1. Terminal Operations

The opening screen of the program is adequate to perform teletype terminal communications over the currently selected communications port, once properly initialized. The initialization commands are fourd in the Activate Master subscreens.

2. Remote Login

The Slave microcomputer may be operated as though the Master operator is entering commands from its keyboard and observing the results on its display. These functions are remoted to the Master screen.

To log in to the Slave, select Activate Master from the main menu and then select options from the secondary menu to establish the correct baud rate, parity, for the port connected to the desired Slave and to connect the port. The default settings are usually satisfactory once the network is established. The Master cannot reset the Slave's port parameters remotely. Once the port is connected, select Remote Login from the Activate Master menu. After a moment for the exchange of command and response, the Slave's local directory will be displayed. From this point, any MS-DOS command or program entered at the Master may be run on the Slave and the output will be displayed at the Master.

3. Initialize Port, Connect Port, Disconnect Port

These commands are used to set the communications port settings, and to establish a link to the attached Slave microcomputer. Both the Slave and Master microcomputers must be set up at the same serial port parameters to communicate. To change to a different Slave (port), either first disconnect the current port and connect the desired port, or simply connect the new port.

4. Equipment Status

This command will return the Slave configuration on the Master screen. The number of disk drives, communications ports, and available memory is displayed.

5. ZCOPY

These commands allow file transfers from or to the connected Slave. Upon activation, the program will prompt for the file name to be sent or received. If the copy will result in another file of the same name being overwritten, confirmation will be asked. The Master will display the Slave's ZCOPY program output after the transfer is complete. This is useful if an error occurs.

6. Reset Remote

This command is useful if the Slave was operating satisfactorily and now appears unresponsive. It aborts any protocol transfer in progress and restores flow control the command receive mode.

7. Exit (ESC)

This exits the Activate Master environment. All communications port selections remain intact.

D. COMMAND SUMMARY

The remaining commands accessed from the main screen are:

1. Alt-A Change drive & path

This command changes the current disk drive and path for file transfers or directory operations. It also determines the starting directory for a DOS shell.

2. Alt-B Send a Break signal

This command sends an RS-232C break signal over the currently selected communications port.

3. Alt-C Update Config File

This command allows the operator to display the current program configuration parameters as found in the file DISTRIB.CFG, in the current directory. An error indication is given if the file is not found. The operator can select any of the displayed parameters to change, and a range of options is displayed. Default settings for the communications ports, the modem dialing prefix, and screen color settings are provided.

4. Alt-D Dialing Directory

This command allows the operator to dial a telephone number from a list of stored numbers, or a number entered manually from the keyboard. This command assumes a Hayes compatible modem.

5. Alt-E Local echo toggle

Intended for terminal operations, this command sets a half duplex toggle to display transmitted as well as received commands if the remote terminal does not echo received characters.

6. Alt-F Change DC params

This command allows the operator to set the baud rate, word length, parity and stop bits for the currently selected communications port, to override the configuration settings.

7. Alt-G Show disk directory

This command displays the local disk directory, in MS-DOS standard or /w formats.

8. Alt-H Hang up phone

This command tells the modem to disconnect the telephone line.

9. Alt-L DOS Shell

This command executes a secondary copy of the MS-DOS command processor to allow the operator to utilize the operating system without terminating the distributing processing program.

10. Alt-M Activate Master

This command opens a second set of commands to command the Slave processor. These include:

Initialize port
Connect to current port
Disconnect current port
ZCOPY file to remote
ZCOPY file from remote
Get machine status
Login to remote machine
Reset remote server

a. Initialize Port

This command allows the operator to select the current port parameters from a menu of options, ranging from 110 baud to 38,400 baud.

b. Connect to Current Port

This command allows the operator to assign a port (currently COM1 or COM2) as the port for current operations.

c. Disconnect Current Port

This command disables the receive interrupts for the currently selected port.

d. ZCOPY file to remote

This command requests the name of the file to be sent to the Slave, and then invokes a program called ZCOPY to send the file at the maximum data rate supported by the communications port. Precautions must be taken if a modem is used, since the modem will dictate the maximum data rate.

e. ZCOPY file from remote

This command requests the name of the file to be received from the Slave, and then invokes a program called ZCOPY to receive the file at the maximum data rate supported by the communications port. Precautions must be taken if a modem is used, since the modem will dictate the maximum data rate.

f. Get machine status

This command allows the Master operator to query the configuration of the connected Slave microcomputer, and displays the number of floppy disk drives, communications ports, and available memory.

g. Login to remote machine

This command returns a prompt from the remote machine on a full screen window at the Master. The operator is then able to send commands to the Slave in much the same manner as from the local operating system prompt. Responses are displayed on the Master screen.

h. Reset remote server

This command is used to resynchronize the Master and Slave computers. It does so by sending a series of CAN characters down the serial communications link to abort any operations in progress and return the Slave to the command mode.

11. PgDn, Alt-R XMODEM Get a file

This command allows the Master to perform a file transfer from an Xmodem compatible remote system. The filename is requested from the operator to assign to the received file.

12. Alt-S Activate Server

This command allows the operator to invoke Slave operations on the local microcomputer, and is useful for systems initialization and setup. Depressing a key while in this mode aborts the Slave operation, but returns the program to the terminal mode.

13. PgUp, Alt-T XMODEM Put a file

This command allows the operator to perform a file transfer to an Xmodem compatible remote system. The filename of the file to be sent is requested from the operator.

14. Alt-X (ESC) Exit emulator

This command halts the program, restores all communications port interrupt vectors, and returns control to the operating system.

E. TERMINATION

1. Slave

Slave operation is terminated by depressing a key. Control returns to the operating system.

2. Master

The Master is terminated by returning to the main menu (terminal screen) and depressing Alt-X. Control returns to the operating system.

APPENDIX B

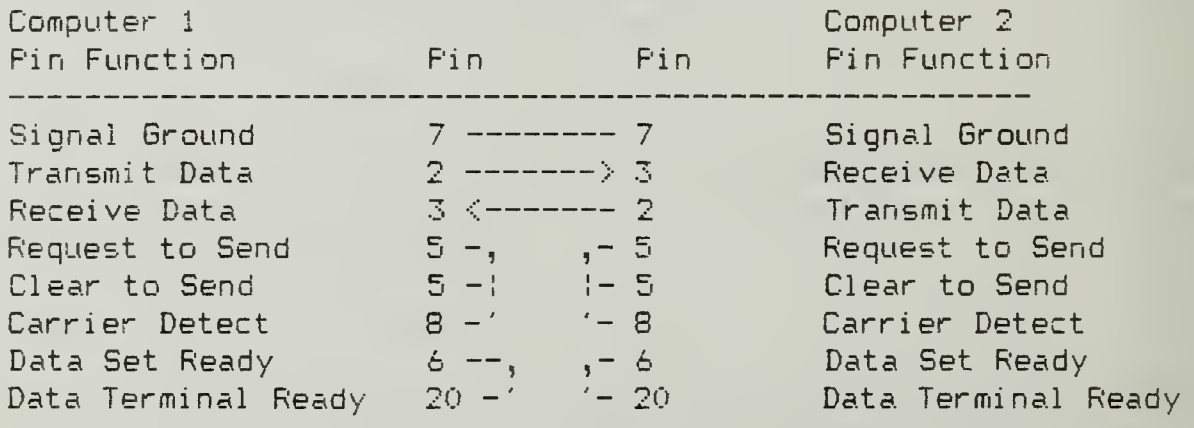
INSTALLATION/PROGRAMMING AIDS

This appendix provides information on the construction of null modem cables for use between the Master and Slave microcomputers, and provides a listing of all procedures and functions found in the distributed processing program. These procedures and functions are sorted alphanumerically within by program or unit.

A. SERIAL PORT CONNECTIONS

The serial connection between computers are the RS-232C communications ports operating at 9600 baud for IBM PC/AT compatible machines and 4800 baud for IBM PC/AT compatibles. The difference is due to some spurious characters noted on the slower machine's display during data transfers. The microcomputers at each end of a single link must be configured for the same speed. The pin connection for the interconnecting cables is shown at Figure B.1. For microcomputers with the nine pin AT style connector, a nine pin to RS-232C 25 pin DB-25 cable is recommended, with a NULL modem in between. Hardware handshaking is turned back in this configuration. The program will operate satisfactorily through a modem if the baud rate is lowered. (Flanders, 1989, p. 252)

FIGURE B.1
SERIAL PORT CONNECTIONS



B. INSTALLATION

Installation may be rapidly accomplished by connecting a null modem cable to COM1 for both the Master and Slave microcomputers. Install a copy of Zcopy.com in the same directory as the Distrib.exe program. The file Distrib.cfg and Distrib.phn should not be resident in this

directory, or the program may initialize the COM1 ports to incompatible settings. Execute the command "Distrib Master" at the MS-DOS prompt of both machines. This should bring both programs up in the terminal mode. Depress the Alt-M (Activate Master) key combination to access the communications port settings and initialize COM1 for 9600 baud, 8 data bits, 1 stop bit and no parity (4800 baud for non - AT IBM PC compatibles). Connect to the COM1 port and press ESC to exit the secondary menu. The Master and Slave should be able to communicate as glass teletypes to each other. If desired, change the default settings for both microcomputers to the desired port parameters by selecting Alt-C (Update Config File). This, when saved, will generate the configuration file for the microcomputer. A similar procedure with Alt-D will allow the creation of the telephone number file if desired. Create an AUTOEXEC.BAT file for the microcomputer(s) designated as Slave and include the command "Distrib Server" to enter the Slave program on power up. A similar file with "Distrib Master" will allow the Master microcomputer to assume that role on power up.

C. UNIT DEPENDENCIES

The following chart (Table B.1) illustrates the the dependencies of the various units in the program, as a guide to the visibility of the data structures, procedures and functions in the interface section of each program module. CRT and DOS are units supplied with the compiler. All programs and units depend on the System unit.

TABLE B.1

UNIT DEPENDENCIES

UNIT/PROGRAM> / DEPENDS ON V	DISIBM	DATACOM	DIRECTOR	ERRORCOD	GENERAL	MISCPACK	PARSER	PRINTER	REDIRECT	SPAWN	SUPPORT	WNDOW	XMODEM
CRT	X			X	X								
DATACOM	X								X		X		
DIRECTOR	X												
DOS	X			X	X								
ERRORCOD	X								X				
GENERAL	X												
MISCPACK	X								X				
PARSER	X												
PRINTER	X												
REDIRECT									X				
SPAWN	X												
SUPPORT	X								X				
WNDOW	X												
XMODEM	X												

C. PROCEDURE/FUNCTION LIST

The following functions and procedures are found within the Distrib program:

1. **Program Distrib**
   a. Change_DC_Parameters

b. Comms_function
c. Dialing_Directory
d. Dial_Phone
e. Dirs
f. Dos_Shell
g. Get_Dial
h. Get_Equip
i. Handle_Alt_Key
j. Hangup
k. Operator_input
l. Operator_message
m. Process_command
n. Reset_remote
o. Remote_Command
p. Rlogin
q. Rx_File
r. Tx_File
s. Save_File
t. TTY

2. **UNIT Datacom**
a. Connected
b. DataComm_Error
c. Disable
d. Disable_Interrupts
e. Enable
f. Enable_Interrupts
g. Establish
h. HexByte
i. HexWord
j. PurgeLine
k. Reset_Chip
l. RS232_Avail
m. RS232_Peek
n. RS232_In
o. RS232_Init
p. RS232_ISR1
q. RS232_ISR2
r. RS232_Out
s. RS_Break
t. RS_Cleanup
u. RS_Eight_Bits
v. RS_Initialize
w. RS_Restore
x. SelectBitRate
y. SelectFraming
z. SelectParity
aa. SelectWordLength
ab. Send_EOI
ac. Send_String

3. **UNIT Director**
   a. GetAttribut
   b. ShowDir
   c. StandBy
   d. ViewDir
   e. WriteEntry

4. **UNIT ErrorCod**
   a. Extended_Error_Code

5. **UNIT General**
   a. Beep
   b. Cursor_Size
   c. Exchange
   d. FillWord
   e. Get_Time
   f. Max
   g. Min

6. **UNIT Miscpack**
   a. BumpStrUp

7. **UNIT Parser**
   a. argc
   b. argv
   c. Init_parse
   d. Parse
   e. ParseName
   f. Parser_main
   g. Resolve_command

8. **UNIT Redirect**
   a. Close_File_Handle
   b. Duplicate_Handle
   c. Init_Redirect_Unit
   d. Redirect_All_Output
   e. Redirect_Handle
   f. Redirect_Std_Input
   g. Redirect_Std_Error
   h. Redirect_Std_Output
   i. Restore_Std_Error
   j. Restore_Std_Input
   k. Restore_Std_Output
   l. Restore_All_Output
   m. Restore_CRT_Assignments

9. **UNIT Spawn**
   a. Match_Command
   b. Process_intrinsic_command
   c. Run_Local

10. **UNIT Support**
    a. Build_Status_Line
    b. Check_Auxport
    c. Check_Keyboard
    d. Find_Environment
    e. GetEquip
    f. Initialize
    g. Modify_Entry
    h. NoFile
    i. OK
    j. Save_File
    k. Yes

11. **UNIT Wndow**
    a. Build_Borders
    b. Close_Window
    c. Get_Dummy_Screen
    d. Get_Real_Screen
    e. Get_Window
    f. Init_Window_Info
    g. Move_Window
    h. Write_Status
    i. Open_Window
    j. Process_Window_Menu
    k. Restore_Window
    l. Save_Window
    m. SetBackground
    n. SetColor
    o. Special_Processing

12. **UNIT Xmodm**
    a. buf_to_string
    b. Command_Xfer
    c. Get_Buffer
    d. Get_response
    e. ReadAux
    f. Receive_Record
    g. Respond_by_file
    h. Send_CAN;
    i. Send_EOT
    j. Send_String
    k. string_to_buf
    l. Sync_Receive
    m. Send_Record
    n. Sync_Send
    o. Transfer_File
    p. Update_Status
    q. WriteAux
    r. Xmodem_Xfer

APPENDIX C

XMODEM PROTOCOL

The following is an overview of the Xmodem protocol, as described by the author. (Trimble, 1989).

A. MODEM PROTOCOL OVERVIEW 178 lines, 7.5K

1/1/82 by Ward Christensen. I will maintain a master copy of this. Please pass on changes or suggestions via CBBS/Chicago at (312) 545-8086, or by voice at (312) 849-6279.

NOTE: this does not include things which I am not familiar with, such as the CRC option implemented by John Mahr.

Last Rev: (none)

At the request of Rick Mallinak on behalf of the guys at Standard Oil with IBM P.C.s, as well as several previous requests, I finally decided to put my modem protocol into writing. It had been previously formally published only in the AMRAD newsletter.

Table of Contents

1. DEFINITIONS
2. TRANSMISSION MEDIUM LEVEL PROTOCOL
3. MESSAGE BLOCK LEVEL PROTOCOL
4. FILE LEVEL PROTOCOL
5. DATA FLOW EXAMPLE INCLUDING ERROR RECOVERY
6. PROGRAMMING TIPS.

1. Definitions

<soh> 01H
<eot> 04H
<ack> 05H
<nak> 15H
<can> 18H

2. Transmission Medium Level Protocol

Asynchronous, 8 data bits, no parity, one stop bit.

The protocol imposes no restrictions on the contents of the data being transmitted. No control characters are looked for in the 128-byte data messages. Absolutely any kind of data may be sent - binary, ASCII, etc. The protocol has not formally been adopted to a

7-bit environment for the transmission of ASCII-only (or unpacked-hex) data , although it could be simply by having both ends agree to AND the protocol-dependent data with 7F hex before validating it. I specifically am referring to the checksum, and the block numbers and their ones-complement.

Those wishing to maintain compatibility of the CP/M file structure, i.e. to allow modemming ASCII files to or from CP/M systems should follow this data format:

ASCII tabs used (09H); tabs set every 8.
Lines terminated by CR/LF (0DH 0AH)
End-of-file indicated by ^Z, 1AH. (one or more)
Data is variable length, i.e. should be considered a continuous stream of data bytes, broken into 128-byte chunks purely for the purpose of transmission.
A CP/M "peculiarity": If the data ends exactly on a 128-byte boundary, i.e. CR in 127, and LF in 128, a subsequent sector containing the ^Z EOF character(s) is optional, but is preferred. Some utilities or programs still do not handle EOF without ^Zs.
The last block sent is no different from others, i.e. there is no "short block".

3. Message Block Level Protocol

Each block of the transfer looks like:

<SOH><blk #><255-blk #><--128 data bytes--><cksum>

in which:
<SOH> = 01 hex
<blk #> = binary number, starts at 01 increments by 1, and wraps 0FFH to 00H (not to 01)
<255-blk #> = blk # after going thru 8080 "CMA" instr, i.e. each bit complemented in the 8-bit block number.
Formally, this is the "ones complement".
<cksum> = the sum of the data bytes only. Toss any carry.

4. File Level Protocol

a. Common to Both Sender and Receiver

All errors are retried 10 times. For versions running with an operator (i.e. NOT with XMODEM), a message is typed after 10 errors asking the operator whether to "retry or quit". Some versions of the protocol use <can>, ASCII ^X, to cancel transmission. This was never adopted as a standard, as having a single "abort" character makes the transmission susceptible to false termination due to an <ack> <nak> or <soh> being corrupted into a <can> and canceling transmission.

The protocol may be considered "receiver driven", that is, the sender need not automatically re-transmit, although it does in the current implementations.

b. Receive Program Considerations

The receiver has a 10-second timeout. It sends a <nak> every time it times out. The receiver's first timeout, which sends a <nak>, signals the transmitter to start. Optionally, the receiver could send a <nak> immediately, in case the sender was ready. This would save the initial 10 second timeout. However, the receiver MUST continue to timeout every 10 seconds in case the sender wasn't ready.

Once into a receiving a block, the receiver goes into a one-second timeout for each character and the checksum. If the receiver wishes to <nak> a block for any reason (invalid header, timeout receiving data), it must wait for the line to clear. See "programming tips" for ideas Synchronizing: If a valid block number is received, it will be:

(1) The expected one, in which case everything is fine; or

(2) a repeat of the previously received block. This should be considered OK, and only indicates that the receivers <ack> got glitched, and the sender re-transmitted;

(3) any other block number indicates a fatal loss of synchronization, such as the rare case of the sender getting a line-glitch that looked like an <ack>. Abort the transmission, sending a <can>.

c. Sending Program Considerations

While waiting for transmission to begin, the sender has only a single very long timeout, say one minute. In the current protocol, the sender has a 10 second timeout before retrying. I suggest NOT doing this, and letting the protocol be completely receiver-driven. This will be compatible with existing programs.

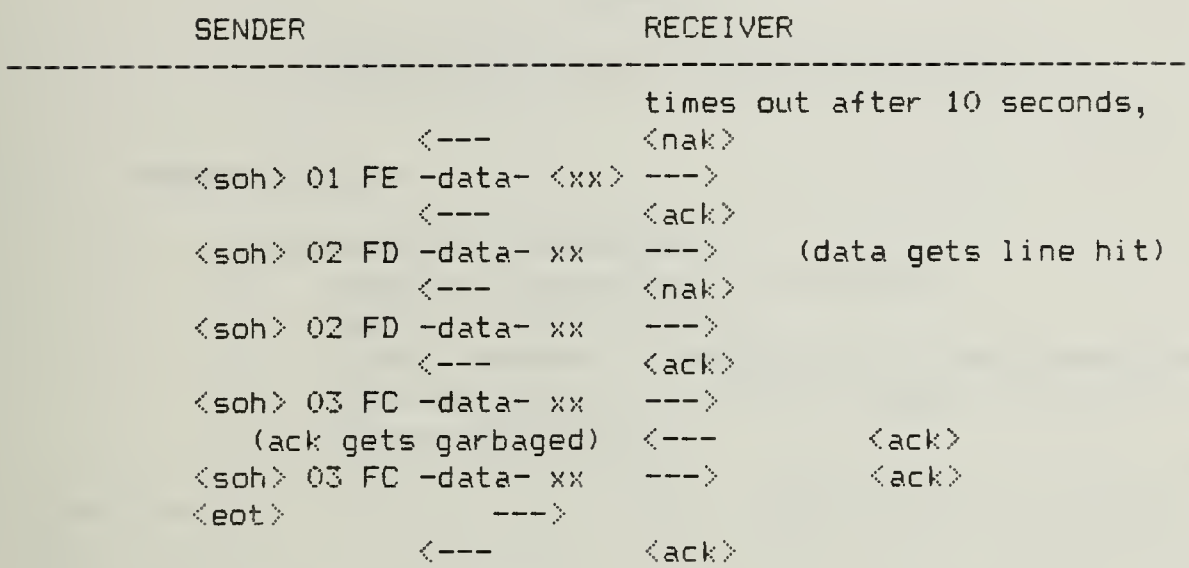
When the sender has no more data, it sends an <eot>, and awaits an <ack>, resending the <eot> if it doesn't get one. Again, the protocol could be receiver-driven, with the sender only having the high-level 1-minute timeout to abort.

5. Data Flow Example Including Error Recovery

Here is a sample of the data flow, sending a 3-block message, which handles the two most common line hits - a garbaged block, and an <ack> reply getting garbaged. <xx> represents the checksum byte.

FIGURE C.1

DATA FLOW EXAMPLE



6. Programming Tips

The character-receive subroutine should be called with a parameter specifying the number of seconds to wait. The receiver should first call it with a time of 10, then <nak> and try again, 10 times.

After receiving the <soh>, the receiver should call the character receive subroutine with a 1-second timeout, for the remainder of the message and the <cksum>. Since they are sent as a continuous stream, timing out of this implies a serious like glitch that caused, say, 127 characters to be seen instead of 128.

When the receiver wishes to <nak>, it should call a "PURGE" subroutine, to wait for the line to clear. Recall the sender tosses any characters in its UART buffer immediately upon completing sending a block, to ensure no glitches were mis-interpreted.

The most common technique is for "PURGE" to call the character receive subroutine, specifying a 1-second timeout, and looping back to PURGE until a timeout occurs. The <nak> is then sent, ensuring the other end will see it.

You may wish to add code recommended by Jonh Mahr to your character receive routine - to set an error flag if the UART shows framing error, or overrun. This will help catch a few more glitches - the most common of which is a hit in the high bits of the byte in two consecutive bytes. The <cksum> comes

out OK since counting in 1-byte produces the same result of adding 80H + 80H as with adding 00H + 00H.

APPENDIX D

MAINTENANCE MANUAL FOR DISTRIB PROGRAM

A. PROGRAM DISTRIB

1. Configuration Information

a. Language - Turbo Pascal Version 4.0
b. Compiler Version - 4.0
c. Target Hardware - IBM PC/AT or close compatible
d. Operating System - Microsoft MS-DOS (Version 3.x)
e. Program Description

Distrib is the main program for both the Master and Slave computers operating in the distributed processing network. The main program loop initializes the window unit, saves the current directory and the current screen image for restoration on program termination, and then calls Initialize in the Support Unit to establish the communications port parameters, screen colors, dialing directory, and other default parameters. The program then examines the command tail following the program name when it was called from the operation system and takes one of the following actions:

(1) Command tail is NIL or "Server". If nothing is specified after the program name, or the word "Server" is found as the first command line parameter, the program assumes it is to operate as a remote Slave or Server and enters a processing loop to wait for a command packet from its communications port. A local screen display is available showing a program version banner and a monitor window showing commands received and responses generated. Local keyboard input after this point will abort the program, reverting the computer to local use.

(2) Command tail is "Master". If the word "Master" is found as the first command line parameter, the program enters the terminal mode through the default communications port and awaits operator action at the local keyboard. If a remote Slave computer is connected, NAK symbols will be displayed periodically as the remote computer awaits a command. A status line is displayed across the 25 line of the screen and HELP is offered to the local operator if the HOME key is depressed. HELP displays a list of available commands to initiate file transfers or run remote programs.

2. Subroutines Contained

a. Dial_Phone
(1) Type: Procedure

(2) Purpose: To dial a selected telephone number on a Hayes compatible modem connected to the modem port.

(3) Description of Parameters: I is the entry number to be dialed that was selected by the user from the Dialing_Directory procedure that follows. Demon_Dial, if TRUE, repeat dials the entry until the modem reports a connection. This procedure changes the COMM port selection stored in the DataCom Unit variable Current_Com to the modem port, and leaves it there.

(4) Subroutines Called:

Flush_Buffer (dumps the receive buffer)
DataCom.Connected
DataCom.RS_Initialize
DataCom.RS_Cleanup
DataCom.RS232_In
DataCom.RS232_Avail
DataCom.Send_String
CRT.ClrEOL
CRT.ClrScr
CRT.Delay
CRT.GoToXY
Wndow.Beep
Wndow.Get_Window
Wndow.Open_Window
Wndow.Close_Window

(5) Process Description

Given the dialing directory entry to dial, the procedure initializes the modem port according to information stored in the dialing entry data structure Support.Phone_Stuff; and sends a string to the modem to dial the number. If repeat dialing is selected, a window is displayed showing the progress of the call.

b. **Get_Dial**

(1) Type: Procedure

(2) Purpose: This procedure allows the operator to select a telephone number to be dialed.

(3) Description of Parameters:

Input: Support.Phone_Menu (the list of available numbers)

Output: The function returns the order of the n'th phone list entry

(4) Subroutines Called:

Wndow.Open_Window
Wndow.Process_Window

(5) Process Description

The procedure calls Open_Window with parameter Phone_Menu from the Support Unit to display a menu of telephone numbers contained in the file DISTRIB.PHN, and allows the operator to select one with a menu bar.

c. **Dialing_Directory**

(1) Type: Procedure

(2) Purpose: To allow the user to dial, modify, add or delete any telephone number entry in the data structure Support.Phone_Stuff.

(3) Description of Parameters: none.

(4) Subroutines Called:

Get_Dial (displays the list of telephone numbers that are available)
CRT.GoToXY
CRT.ClrEOL
CRT.ClrScr
Support.Modify_Entry
Support.OK
System.FreeMem
System.GetMem
System.Move
System.SizeOf
Wndow.Get_Window
Wndow.Open_Window
Wndow.Close_Window

(5) Process Description

This procedure first displays a window allowing the operator to dial, modify, add or delete any number in the data structure Support.Phone_Stuff. If dial is selected, the number is dialed and the program returns to terminal mode. If modify or delete is selected, a list of available names attached to known telephone numbers is displayed for selection. If a number is to be added, a blank parameter table is displayed for data entry. On completion, the operator is offered the opportunity to save the added number to the file DISTRIB.PHN, through a call to Modify_Entry. ESC returns to the terminal mode.

d. Dirs

(1) Type: Procedure

(2) Purpose: To allow the user to display the local disk directory.

(3) Description of Parameters: none.

(4) Subroutines Called:

CRT.GoToXY
CRT.ClrEOL
CRT.ClrScr
DOS.Find_First
DOS.Find_Next
System.ChDir
System.GetDir
System.ReadKey
Wndow.Open_Window
Wndow.Close_Window

(5) Process Description

This procedure prompts the user for a path specification and directory mask similar to that used by the MS-DOS DIR

command and then displays the directory for that specification a screen at a time. Capabilities similar to DIR *.* and DIR *.*/w are provided.

e. Change_DC_Parameters

(1) Type: Procedure

(2) Purpose: To allow the user to select speed, parity, word length and stop bit parameters for the COM port specified by DataCom.Current_Com.

(3) Description of Parameters: DataCom.Current_Com

(4) Subroutines Called:

CRT.ClrScr
DataCom.RS_Initialize
DataCom.RS_Cleanup
Wndow.Open_Window
Wndow.Close_Window
Wndow.Process_Window

(5) Process Description

This procedure offers a selection of parameter combinations for the currently selected COM port and allows the port to be configured accordingly. A menu bar selection is used.

f. Hangup

(1) Type: Procedure

(2) Purpose: To hang up the modem.

(3) Description of Parameters: DataCom.Current_Com

(4) Subroutines Called:

CRT.Delay
DataCom.RS232_In
DataCom.RS232_Avail
DataCom.RS_Initialize
DataCom.RS_Cleanup
DataCom.Send_String

(5) Process Description

This procedure places the modem in command mode and sends a disconnect command string to the Hayes compatible modem connected to the current communications port.

g. Operator_Input

(1) Type: Function

(2) Purpose: To obtain a string input from the operator.

(3) Description of Parameters: Title is a string typed in the Wndow Unit that is to be displayed on the window; Prompt is a string written in the window area specifying what the operator is to enter.

(4) Subroutines Called:

CRT.ClrScr
Wndow.Open_Window
Wndow.Close_Window

(5) Process Description

This function opens a titled window and waits for the operator to type a string. The string is returned as the function result.

h. Operator_Message

(1) Type: Function

(2) Purpose: To inform the operator with a string message, usually of some error condition that is to be temporarily displayed.

(3) Description of Parameters: Title is a string typed in the Wndow Unit that is to be displayed on the window; Message is the string message to be provided to the operator. Note that this function depends on the calling program to close the window.

(4) Subroutines Called:

CRT.ClrScr
Wndow.Open_Window

(5) Process Description

This function opens a titled window and places the message string in the window.

i. Process_Command

(1) Type: Function

(2) Purpose: To operate the computer as a Slave, process all requests to initialize COM ports, transfer files between Master and Slave computers, remotely operate a Slave computer, or reset the connection between computers.

(3) Description of Parameters: The function returns to the calling program an enumerated state variable defined in the Unit Xmodm depending on the successful dispatch of a command to a Slave computer and the receipt of the response, or an indication that the local operator has aborted the operation by pressing a key. The keypressed indication is typically all that is of interest, since the function normally called repeatedly.

(4) Subroutines Called:

CRT.ClrScr
CRT.GoToXY
System.ReadKey
Wndow.Open_Window
Wndow.Close_Window
Wndow.Get_Window
Wndow.Process_Window
Xmodm.Buf_to_String
Xmodm.Command_Xfer
Xmodm.Send_CAN
Xmodm.String_to_buf
Xmodm.Respond_by_file

(5) Process Description

The initial state of the communications link is from Master to Slave (this process). This function opens a small status window indicating whether it is awaiting a remote command,

parsing a received command for local execution, or completing the command execution. It does so in this sequence: First, a loop is entered that repeatedly calls the function Xmodm.Command_Xfer. On successful receipt (status = Rx_done), the command is converted from an Xmodem packet into a string and passed to Parser.Parser_main for execution. The communications link also switches direction, with the Master expected the Slave to initiate Xmodem packet transmissions. This procedure returns any error indication from the locally executed procedure or spawned program as a string in the variable Error_Msg, along with a typed variable Errtype indicating whether the response is a file (for program results or output) or a simple string variable or nothing at all (NULL string). Errtype is used in a following CASE construct to send the file specified by a complete drive and path specification in Error_Msg back to the Master computer, or to forward Error_Msg as a packetized string utilizing the Transmit option of Xmodm.Command_Xfer. Similarly, this procedure returns any output from the locally executed procedure or spawned program as a string in the variable Response, along with a typed variable Restype indicating whether the response is a file (for program results or output) or a simple string variable or nothing at all (NULL string). Restype is used in a following CASE construct to send the file specified by a complete drive and path specification in Response back to the Master computer, or to forward Response as a packetized string utilizing the Transmit option of Xmodm.Command_Xfer. The Master computer expects a response of this type over the communications line when it detects the successful command transfer. Note that the normal exit condition for the Command_Xfer loops throughout this function is Rx_Done or Tx_Done. The Master computer will continue to display responses from the Slave until a CAN character is received. At this point, the function returns with the last valid status of the Command_Xfer function, and the communications link again switches to the beginning state, with the Slave waiting on transmissions from the Master. Error indications other than that in Error_Msg short circuit the program execution through this function, send a CAN character to the Master, return the communications link to its initial state, and leave the function with an error status.

j. Reset_Remote

(1) Type: Procedure

(2) Purpose: This subprocedure of the Comms_Function allows the operator to recover control of the Slave computer if synchronization is lost over the communications link.

(3) Description of Parameters: None.

(4) Subroutines Called:

Update.Status (local to Comms_Function)

Xmodm.Send_CAN

(5) Process Description

This procedure sends four CAN characters out on the communications link to the Slave. The Process_Command function (described above) is sensitive to the receipt of CAN characters and will exit the function early with an error status. The calling program

simply loops into the Process_Command function again and awaits a command.

k. **Remote_Command**

(1) Type: Function

(2) Purpose: This subfunction of Comms_Function function accomplishes one cycle of a Master to Slave command and response over the communications port.

(3) Description of Parameters: The function is entered with a string containing the command to be executed. The function returns to the calling program an enumerated state variable defined in the Unit Xmodm depending on the successful dispatch and execution of a command by the Slave computer, or an indication that a local operator has aborted the sequence by depressing a key. The keypressed indication is typically all that is of interest, since the function normally called repeatedly.

(4) Subroutines Called:

System.ReadKey
Xmodm.Command_Xfer
Xmodm.String_to_buf

(5) Process Description

This function is currently called by Get_Equip to perform a single command cycle; or Rlogin to repeatedly cycle and allow the operator to remotely operate the Slave computer from the Master keyboard in a manner similar to the DOS prompt. It does so in this sequence: First, a loop is entered that repeatedly calls the function Xmodm.Command_Xfer to pass the command string to the Slave. On successful transmission (status = Tx_done), function Xmodem.Get_Response displays the packetized response from the Slave on the Master monitor window. The Master continues to display responses from the Slave until the Slave sends a CAN character, indicating completion of the all responses, or the Master operator depresses a key to break the cycle. At this point, the function returns with a boolean indication of the success of the transfer
(TRUE = success, FALSE for any keypress during the cycle).

l. **Rlogin**

(1) Type: Procedure

(2) Purpose: This subprocedure of the Comms_Function function cycles the Remote_Command function and allows operator input of commands to the Slave until aborted by the operator.

(3) Description of Parameters: None.

(4) Subroutines Called:

Update.Status (local to Rlogin)
CRT.ClrScr
Distrib.Remote_Command
Distrib.Reset_Remote
Distrib.Operator_Input
Wndow.Open_Window
Wndow.Close_Window
Wndow.Get_Window

(5) Process Description

At the beginning, this procedure opens a full screen window to display all responses from the Slave in much the same way a local operator would view them. The procedure then calls Remote_Command initially with a command string requesting a prompt from the remote system so that the operator can determine the current directory of the Slave. If that succeeds, the Master operator is prompted for a command to send to the Slave by Operator_Input. Remote processing may be terminated by entering an exclamation point ("!") whereupon the operator is asked to confirm the termination. Remote processing also terminates if Rlogin returns a FALSE result. On exit, the procedure closes the monitor window and exits.

m. Rx_File

(1) Type: Procedure

(2) Purpose: This subprocedure of the Comms_Function function initiates a file transfer from the Slave to the Master by using an adaptive file transfer program, Zcopy.

(3) Description of Parameters: None.

(4) Subroutines Called:

Update.Status (local to Rlogin)
CRT.ClrScr
Distrib.Remote_Command
Distrib.Operator_Input
System.Exec
Wndow.Open_Window
Wndow.Close_Window
Wndow.Get_Window
Xmodm.String_to_buf

(5) Process Description

This procedure opens a full screen window to display the operation of the Zcopy file transfer program, and prompts the operator for the name of the file to receive. This file is assumed to be in the current directory of the Slave unless a full path is specified. Once the file name is obtained, a command string is assembled to send to the Slave to initiate the transfer. The procedure is terminated if the command transfer is interfered with by a keypress at the Master. Once the Slave acknowledges receipt of the command, the Master initiates the Zcopy program locally, using a different format to operate as a server under the temporary control of the Slave. The operator is provided prompting information from the Zcopy program in a full screen window if a file must be overwritten or Zcopy synchronization is not achieved. Once completed or terminated, the procedure displays the Zcopy display output from the Slave computer for error diagnostics (if needed), closes all opened windows and exits.

n. Tx_File

(1) Type: Procedure

(2) Purpose: This subprocedure of the Comms_Function function initiates a file transfer from the Master to the Slave by using an adaptive file transfer program, Zcopy.

(3) Description of Parameters: None.

(4) Subroutines Called:

Update.Status (local to Rlogin)
CRT.ClrScr
Distrib.Remote_Command
Distrib.Operator_Input
System.Exec
Wndow.Open_Window
Wndow.Close_Window
Wndow.Get_Window
Xmodm.String_to_buf

(5) Process Description

This procedure opens a full screen window to display the operation of the Zcopy file transfer program, and prompts the operator for the name of the file to transmit. This file is assumed to be in the current directory of the Master unless a full path is specified. Once the file name is obtained, a command string is assembled to send to the Slave to initiate the transfer. The procedure is terminated if the command transfer is interfered with by a keypress at the Master. Once the Slave acknowledges receipt of the command, the Master initiates the Zcopy program locally, operating as a file transfer master with the Slave operating as a Slave. The operator is provided prompting information from the Zcopy program in a full screen window if a file must be overwritten or Zcopy synchronization is not achieved. Once completed or terminated, the procedure displays the Zcopy display output from the Slave computer for error diagnostics (if needed), closes all opened windows and exits.

o. **Get_Equip**

(1) Type: Procedure

(2) Purpose: This subprocedure of the Comms_Function function displays the communications port and floppy disk configuration of the Slave computer.

(3) Description of Parameters: None.

(4) Subroutines Called:

Update.Status (local to Rlogin)
CRT.ClrScr
Distrib.Remote_Command
Distrib.Operator_Input
Wndow.Open_Window
Wndow.Close_Window

(5) Process Description

Utilizing the Remote_Command function, this procedure dispatches the command string "Equip" to the Slave, which is processed in the Slave program to obtain BIOS information via BIOS call $11. On exit, the procedure closes the remote monitor window and exits.

p. **Comms_Function**

(1) Type: Function

(2) Purpose: To process operator requests to initialize COM ports, transfer files between Master and Slave computers, remotely operate a Slave computer, or reset the connection between computers.

(3) Description of Parameters: The function returns to the calling program an enumerated state variable defined in the Unit Xmodm depending on the successful dispatch of a command to a Slave computer and the receipt of the response, or an indication that the local operator has aborted the operation by pressing a key. The keypressed indication normally allows the operator to make another selection or to leave this function.

(4) Subroutines Called:

Update.Status (for local display of the system state)

CRT.ClrScr
CRT.GoToXY
Distrib.Remote_Command
Distrib.Rlogin
Distrib.Rx_File
Distrib.Tx_File
Distrib.Get_Equip
System.ReadKey
Wndow.Open_Window
Wndow.Close_Window
Wndow.Get_Window
Wndow.Process_Window
Xmodm.Buf_to_String
Xmodm.Command_Xfer
Xmodm.Send_CAN
Xmodm.String_to_buf
Xmodm.Respond_by_file

(5) Process Description

This function opens a window showing the parameters for the current communications port, and a second window to allow the operator to select one of the following functions: Initialize a port, change to a different port and enable the receive interrupts, disable a receive interrupts for a port, send a file to the Slave computer, receive a file from the Slave, obtain the port and disk configuration of the Slave, operate the Slave remotely, reset the current Xmodem link, and leave the function. It does so by calling one of the following procedures or functions local to Comms_Function by a CASE selection: Distrib.Remote_Command, Distrib.Rlogin, Distrib.Rx_File, Distrib.Tx_File, Distrib.Get_Equip.

q. **DOS_Shell**

(1) Type: Procedure

(2) Purpose: This procedure spawns a copy of the MS-DOS command processor to allow the operator of the Master computer to perform DOS functions while retaining the control program. Control is returned to the Master program on exiting the secondary processor.

(3) Description of Parameters: None.

(4) Subroutines Called:

CRT.ClrEOL
CRT.ClrScr
CRT.Delay
Distrib.Find_Environment
Support.OK
System.ChDir
System.GetDir
System.Exec
Wndow.Open_Window
Wndow.Close_Window

(5) Process Description

The procedure first locates a copy of the DOS command processor by finding the "COMSPEC=" path specification in the current environment. This is established on startup of the computer and is normally passed down to the application program for its use. Once this complete file specification is obtained, the operator is informed that the DOS shell will be activated and a full screen window is opened to save the current screen. When the operator terminates the secondary command processor by entering "EXIT" at the prompt, the procedure restores the original disk drive and directory, notes any DOS errors returned, and returns to the terminal screen. If the COMSPEC environment parameter cannot be found, the procedure informs the operator, obtains acknowledgment, and exits.

r. **Handle_ALT_Key**

(1) Type: Procedure

(2) Purpose: This procedure dispatches the program to a particular function selected by the operator as an ALT-key. A help display is also provided as offered on the status line.

(3) Description of Parameters: B is the high order byte read from the keyboard and is used as a CASE selector

(4) Subroutines Called:

CRT.ClrEOL
CRT.ClrScr
CRT.Delay
DataCom.RS_Break
Distrib.Change_DC_Parameters
Distrib.Comms_Function
Distrib.Dialing_Directory
Distrib.Dirs
Distrib.DOS_Shell
Distrib.Hangup
Distrib.Handle_ALT_Key (the procedure calls itself after processing the help menu)
Support.Build_Status_Line
Support.Modify_Entry
Support.OK
System.ChDir
Wndow.Beep

Wndow.Close_Window
Wndow.Open_Window
Wndow.Process_Window_Menu
Xmodm.Transfer_File

(5) Process Description
The functions offered by this procedure are:
Alt-A: Change Drive and Path
Alt-B: Send a Break signal out of the current COM port
Alt-C: Clear the screen
Alt-D: Dial a telephone number and connect by modem
Alt-E: Toggle the local Echo for half duplex communications
Alt-F: Change the default communications parameters
Alt-G: Show the current directory
Alt-H: Hang up the modem
Alt-L: Open the DOS Shell
Alt-M: Activate the Master
Alt-P: Activate the Master
Alt-R, PgDn: Receive a file via Xmodem
Alt_S: Activate the Server
Alt-T, PgUp: Transmit a file via Xmodem
Alt-X: Terminate the program
Home: Display a help screen of these commands and allow selection by menu bar

s. TTY
(1) Type: Procedure
(2) Purpose: This procedure provides a teletype emulation augmented by ANSI control functions.
(3) Description of Parameters: ANSI = TRUE indicates the procedure acts as an ANSI terminal emulator.
(4) Subroutines Called:
WriteLF (process a line feed)
DOS Interrupt $10 (Video Display)
CRT.ClrScr
CRT.Delay
Wndow.Open_Window
Wndow.Close_Window
Support.OK
System.ChDir
System.GetDir
System.Exec
(5) Process Description
The procedure filters characters generated by the keyboard and arriving from the communications port in the terminal mode to emulate an ANSI terminal. ALT-key combinations are intercepted from the keyboard and processed by Handle_ALT_Key.

APPENDIX E

MAINTENANCE MANUAL FOR UNIT DATACOM

A. UNIT DATACOM

1. Configuration Information

a. **Language - Turbo Pascal Version 4.0**

b. **Compiler Version - 4.0**

c. **Target Hardware - IBM PC/AT or close compatible**

d. **Operating System - Microsoft MS-DOS (Version 3.x)**

e. **Program Description**

Provides all procedures and functions needed to initialize the computer serial communications ports, enable and disable receive interrupts, provide buffered reception of characters, clear the receive buffer(s), send or receive bytes through the ports, send a BREAK signal over the RS-232 port, and nondestructively read the receive buffer(s). Supports Unit Xmodem and the terminal portion of Distrib. The currently selected communications port is contained in public variable Current_Com.

2. Subroutines Contained

a. **Disable_Interrupts**

(1) Type: Procedure

(2) Purpose: To permit a Pascal procedure to disable system interrupts.

(3) Description of Parameters:

Input: None.

Output: System interrupts are disabled.

(4) Subroutines Called:

Inline assembly.

(5) Process Description

The assembly instruction to mask off interrupts at the CPU is inserted into the code stream at compile time.

b. **Enable_Interrupts**

(1) Type: Procedure

(2) Purpose: To permit a Pascal procedure to enable system interrupts.

(3) Description of Parameters:

Input: None.

Output: System interrupts are enabled.

(4) Subroutines Called:

Inline assembly.

(5) Process Description

The assembly instruction to unmask interrupts at the CPU is inserted into the code stream at compile time.

c. RS232_ISR1

(1) Type: Procedure

(2) Purpose: The interrupt service routine for communications port one.

(3) Description of Parameters:

Input: An interrupt vector call initiated from communications port one.

Output: The received character is placed in a buffer.

(4) Subroutines Called:

DataCom.DisableInterrupts
DataCom.EnableInterrupts
System.Port

(5) Process Description

System interrupts are temporarily turned off to service this interrupt. The UART Line Status Register for communications port one is read to record any error indications, then the Receive Buffer Register is read to place the character in the receive buffer. The buffer tail pointer is advanced and an End of Interrupt command is sent to the Programmable Interrupt Controller to signal the end of the interrupt service call.

d. RS232_ISR2

(1) Type: Procedure

(2) Purpose: The interrupt service routine for communications port two.

(3) Description of Parameters:

Input: An interrupt vector call initiated from communications port two.

Output: The received character is placed in a buffer.

(4) Subroutines Called:

DataCom.DisableInterrupts
DataCom.EnableInterrupts
System.Port

(5) Process Description

System interrupts are temporarily turned off to service this interrupt. The UART Line Status Register for communications port two is read to record any error indications, then the Receive Buffer Register is read to place the character in the receive buffer. The buffer tail pointer is advanced and an End of Interrupt command is sent to the Programmable Interrupt Controller to signal the end of the interrupt service call.

e. RS_Break

(1) Type: Procedure

(2) Purpose: To instruct the UART on the currently selected communications port to send and RS-232 BREAK signal.

(3) Description of Parameters:

Input: Current_Com (public)

Output: A break signal is generated on the currently selected communications port.

(4) Subroutines Called:

CRT.Delay

System.Port

(5) Process Description

This process ORs the current contents of the UART Line Control Register with constant LCR_BREAK to instruct the UART to send a constant space on the output line. A UART receiving this will set its LSR_BREAK to signal a BREAK received. After a delay of about 1/5 second, the line is restored.

f. RS232_Avail

(1) Type: Function

(2) Purpose: Informs the calling program that received characters are available to be read from the current communications port.

(3) Description of Parameters:

Input: Current_Com (public)

Output: TRUE if characters available, FALSE otherwise.

(4) Subroutines Called: None.

(5) Process Description

The buffer pointers RS_Buf_Head [Current_Com] and RS_Buf_Tail [Current_Com] will be equal if the buffer is empty, the function returns the result of this test.

g. Purgeline

(1) Type: Procedure

(2) Purpose: Dump the receive buffer and clear the UART receive registers. Used to clear the communications line prior to an Xmodem packet reception (Christensen, 1982, p. 3).

(3) Description of Parameters:

Input: Current_Com (public)

Output: The internal buffers are cleared.

(4) Subroutines Called:

System.Port

(5) Process Description

The buffer pointers RS_Buf_Head [Current_Com] and RS_Buf_Tail [Current_Com] are both set to their initial conditions (zero) and the UART receive register is read to reset any pending receive interrupt.

h. Connected

(1) Type: Function

(2) Purpose: Returns TRUE if the currently selected communications port is receiving a hardware handshaking signal, indicating the presence of a modem or a directly connected computer.

(3) Description of Parameters:
Input: Current_Com (public)
Output: TRUE if connected, FALSE otherwise.

(4) Subroutines Called:
System.Port

(5) Process Description
The UART Modem Status Register is read to detect the presence of Data Carrier Detect. This line is normally TRUE if a modem or computer is connected .

i. **RS_232_Peek**

(1) Type: Function

(2) Purpose: Nondestructive read of the receive buffer of the current communications port. Used to assist Xmodem synchronization in Unit Xmodm.

(3) Description of Parameters:
Input: Current_Com (public)
Output: The next available received character.

(4) Subroutines Called:
CRT.Delay

(5) Process Description
The receive buffer pointers are compared for the currently selected communications port. If unequal, a character is available. If equal, a short delay is run and the test is repeated. When a character is available, it is returned from this function without disturbing the pointers.

j. **RS_232_In**

(1) Type: Function

(2) Purpose: Read the next character from the the receive buffer of the current communications port. Used for all port reads.

(3) Description of Parameters:
Input: Current_Com (public)
Output: The next available received character.

(4) Subroutines Called:
CRT.Delay

(5) Process Description
The receive buffer pointers are compared for the currently selected communications port. If unequal, a character is available. If equal, a short delay is run and the test is repeated. When a character is available, it is returned from this function and the buffer head pointer is advanced.

k. **RS_232_Out**

(1) Type: Procedure.

(2) Purpose: Send a character out of the currently selected communications port. Used for all port writes.

(3) Description of Parameters:

Input: Current_Com (public); and Param, the character to be sent.

Output: The character is sent to the port. RS_Error (public) is cleared.

(4) Subroutines Called:

CRT.Delay

System.Port

(5) Process Description

The UART Line Status Register is checked on the currently selected communications port to see if the last character has been sent. If not, a short delay is runand the test is repeated. When the buffer is clear, the port Modem Control Register Request To Send and OUT2 lines are set to insure the hardware is prepared to send a character. Next, the corresponding Data Set Ready and Clear To Send status lines are checked and short delays run until they are true, if the options are selected. Last, the character is sent to the port and the error flag is cleared.

l. **Enable**

(1) Type: Procedure

(2) Purpose: Enable receive interrupts for a communications port.

(3) Description of Parameters:

Input: IRQ.

Output: The proper Interrupt Mask Bit in the Programmable Interrupt Controller is cleared for the communications port.

(4) Subroutines Called:

System.Port

(5) Process Description

The procedure masks off the selected bit in the PIC Interrupt Mask Register.

m. **Disable**

(1) Type: Procedure

(2) Purpose: Disable receive interrupts for a communications port.

(3) Description of Parameters:

Input: IRQ.

Output: The proper Interrupt Mask Bit in the Programmable Interrupt Controller is set for the communications port.

(4) Subroutines Called:

System.Port

(5) Process Description

The procedure sets the selected bit in the PIC Interrupt Mask Register.

n. **Establish**

(1) Type: Procedure

(2) Purpose: Enable the Data Terminal Ready, OUT2 and Request To Send handshaking bits on the selected communications port.

(3) Description of Parameters:

Input: Com, the communications port to be enabled.

Output: The appropriate lines are set.

(4) Subroutines Called:

System.Port

(5) Process Description

The OR combination of the Data Terminal Ready, OUT2 and Request To Send handshaking bits are set.

o. **Send_EOI**

(1) Type: Procedure

(2) Purpose: Sends a specific End Of Interrupt command to the 8259 Programmable Interrupt Controller to indicate that a particular interrupt has been serviced.

(3) Description of Parameters:

Input: IRQ, the interrupt serviced.

Output: The Interrupt Service Register bit for the specific interrupt is cleared.

(4) Subroutines Called:

System.Port

(5) Process Description

The bit for specific interrupt is OR'd with $60 and sent to the PIC.

p. **Reset_Chip**

(1) Type: Procedure

(2) Purpose: To shut down a communications port.

(3) Description of Parameters:

Input: Com, the port to be disabled.

Output: The port is cleared, all handshaking lines are cleared, and interrupts are disabled on the UART.

(4) Subroutines Called:

System.UpCase

System.Length

(5) Process Description

The UART Line Status Register is read repeatedly to clear all receive buffers. The system interrupts are disabled to prevent further interrupts from this port. The interrupts from the UART are disabled, and all port handshaking lines are dropped. The Programmable Interrupt Controller interrupt enable line for this port is reset. System interrupts are then restored.

q. **RS232_Init**

(1) Type: Procedure

(2) Purpose: Initialize the selected communications port.

(3) Description of Parameters:

Input: COM, the port to be initialized; and Params, the port parameter word.

Output: The port is initialized.

(4) Subroutines Called:

DOS.Intr($14), the communications port service interrupt.

(5) Process Description

Com is adjusted to satisfy the requirements of Intr($14) and register DX loaded with the communications port to be initialized. The packed word, Params, is loaded into register AX and the interrupt is called.

r. **SelectBitRate**

(1) Type: Procedure

(2) Purpose: Initialize the selected communications port.

(3) Description of Parameters:

Input: COM, the port to be initialized; and Speed, the data rate for the port.

Output: The port is initialized.

(4) Subroutines Called:

System.Port

System.Portw

(5) Process Description

The communications port identified by Com is accessed and its Divisor Latch Access Bit is set to access the bit rate registers. The Speed parameter is mapped into a 16 bit control word and placed in the UART Divisor Latch. The Divisor Latch Access Bit is then cleared and the port is allowed to settle. The current baud rate setting is stored in the port initialization record for later reference.

s. **SelectWordLength**

(1) Type: Procedure

(2) Purpose: Initialize the selected communications port.

(3) Description of Parameters:

Input: COM, the port to be initialized; and Length, the word length for the port.

Output: The port is initialized.

(4) Subroutines Called:

System.Port

System.Portw

(5) Process Description

The Speed parameter is mapped into an 8 bit control word and placed in the UART Line Control Register. The current length setting is stored in the port initialization record for later reference.

t. **SelectFraming**

(1) Type: Procedure

(2) Purpose: Initialize the selected communications port.

(3) Description of Parameters:

Input: COM, the port to be initialized; and Stop, the number of stop bits for the port.

Output: The port is initialized.

(4) Subroutines Called:

System.Port

System.Portw

(5) Process Description

The Stop parameter is mapped into an 8 bit control word and placed in the UART Line Control Register. The current stop setting is stored in the port initialization record for later reference.

u. SelectParity

(1) Type: Procedure

(2) Purpose: Initialize the selected communications port.

(3) Description of Parameters:

Input: COM, the port to be initialized; and P, the type of parity for the port.

Output: The port is initialized.

(4) Subroutines Called:

System.Port

System.Portw

(5) Process Description

The P parameter is mapped into an 8 bit control word and placed in the UART Line Control Register. The current stop parity is stored in the port initialization record for later reference.

v. Send_String

(1) Type: Procedure.

(2) Purpose: To send an ASCII string of characters out the currently selected COM port. Typically used to send command strings to a modem.

(3) Description of Parameters:

Input: S, the string to be sent.

Output: The string is sent out the currently selected COM port.

(4) Subroutines Called:

DataCom.RS232_Out

System.Length

(5) Process Description

The string is treated as an indexed array of characters, and each character is sent to procedure RS232_Out in turn.

w. RS_Initialize

(1) Type: Procedure.

(2) Purpose: To set the communications port to the input parameters.

(3) Description of Parameters:

Input: Com, the port to be initialized; Speed, an enumerated type ranging from 110 baud to 9600 baud; Parity, an enumerated type specifying No Parity, Odd, Even, or Don't Care; The number of stop bits (1 or 2) and the length of the character word (5, 6, 7 or 8 bits).

(3) Output: The communications port is initialized.

(4) Subroutines Called:

DOS.Intr(#14) (BIOS communications port service)
DOS.SetIntVec
System.Port

(5) Process Description

Com and the input parameters are adjusted for the BIOS call. The BIOS call initializes the port, however, it also disables UART receive interrupts. These are enabled separately and the UART Divisor Latch Access Bit is cleared to insure that further writes to the UART will set the proper registers. The UART is recycled and the hardware handshaking lines set. Receive interrupts are enabled at the UART, and the Programmable Interrupt Controller is enabled for the current communications port. The proper interrupt vector for this port is set to point to our interrupt service routine. The settings stored in data structure CommPort [Com] for future reference by RS_Restore.

x. **RS_Restore**

(1) Type: Procedure/Function

(2) Purpose: Restores the parameters of the communications port to the settings stored in data structure CommPort [Com]. Used after a child process is spawned to recover communications port operations.

(3) Description of Parameters:

Input: Com, the communications port to be restored

Output: The selected port is restored.

(4) Subroutines Called:

DataCom.RS_Initialize

(5) Process Description

Com and the parameters stored in ComPort [Com] are used to call RS_Initialize.

y. **RS_Eight_Bits**

(1) Type: Procedure

(2) Purpose: To set the current communications port to eight data bits for Xmodm transfers.

(3) Description of Parameters:

Input: Current_Com (public)

Output: The communications port is set for eight data bits.

(4) Subroutines Called:

System.Port

(5) Process Description

The UART Line Control Register is ORed with $03, setting the number of data bits to eight.

z. **RS_Cleanup**

(1) Type: Procedure

(2) Purpose: Disables interrupts for the current communications port at the Programmable Interrupt Controller.

(3) Description of Parameters:

Input: Current_Com (public)

Output: The PIC is reset for this interrupt.

(4) Subroutines Called:

System.Port

(5) Process Description

The interrupt mask bit for the current communications port is set.

aa. **HexByte**

(1) Type: Function

(2) Purpose: Converts a byte into its hexadecimal string equivalent for the Unit Exit procedure.

(3) Description of Parameters:

Input: B, the byte to be converted.

Output: A string of length two.

(4) Subroutines Called: None.

(5) Process Description

The byte is first shifted right four bits to consider only the high order bits, and a character indexed from the hexadecimal sequence HexDigit. This is concatenated with the character produced by indexing HexDigit by the low order four bits of B to form the two digit hex equivalent.

ab. **HexWord**

(1) Type: Function

(2) Purpose: Converts a word into its hexadecimal string equivalent for the Unit Exit procedure.

(3) Description of Parameters:

Input: I, the word to be converted.

Output: A string of length four.

(4) Subroutines Called:

DataCom.HexByte.

System.Hi

System.Lo

(5) Process Description

HexByte is called with both the high and low order bytes of the word, and the resulting function results concatenated to produce a four digit hex equivalent.

ac. **DataComm_Error**

(1) Type: Procedure

(2) Purpose: Provides a robust means of handling program faults while still insuring that interrupts are restored.

(3) Description of Parameters:

Input: System variables ExitCode, a word that gives an indicaton of why program termination occured; and ErrorAddr, a pointer containing a runtime error address if nonzero;

Output: The procedure writes any error messages desired to the display and resets any interrupt vectors to their state before program execution.

(4) Subroutines Called:

Dos.SetIntVec
System.Assign
System.Rewrite
DataCom.Hex

(5) Process Description

This procedure is chained in to the normal exit processing that the compiler installs for the unit and the unit initialization code. It must be compiled using the Far Call model to be accessible by the program runtime library. The procedure first checks ExitCode and ErrorAddr for abmormal program termination and sets Output to the standard file output for display to allow error message display. The procedure then reports a USER BREAK or runtime error and address if applicable. The program then insures any interrupt vectors are restored and the communications ports are shut down. The Programmable Interrupt Controller Interrupt Mask Register is restored from a saved location. Finally, the original exit code for this unit is restored from a saved location for use by the runtime system (TurboPascal Owner' Handbook, pp. 369-370).

ad. DataCom Unit Initialization Code

(1) Type: Procedure

(2) Purpose: Initializes the Unit, stores critical vectors and registers for restoration on program termination.

(3) Description of Parameters:

Input: System variables ExitProc, a pointer that gives the address of the DataCom unit exit procedure in the runtime library.

Output: The procedure DataComm_Error is linked in before the runtime exit procedure to accomplish an orderly termination of the unit.

(4) Subroutines Called:

Dos.GetIntVec
System.Port

(5) Process Description

The procedure first sets CRT.CheckBreak to TRUE to allow user termination of the program. A pointer to the runtime exit procedure is saved, as well as the current settings for the Programmable Interrupt Controller Interrupt Mask Register for restoration on exit. GetIntVec is used to save the current interrupt vectors for communications ports one and two for restoration on exit. The communications port buffers are cleared, and the unit supplied exit

procedure DataComm_Error is linked in to the runtime system (TurboPascal Owner' Handbook, pp. 369-370). Finally, the two communications ports are assigned default parameters, although not initialized at this time.

APPENDIX F

MAINTENANCE MANUAL FOR UNIT DIRECTOR

A. UNIT DIRECTOR

1. Configuration Information

a. Language - Turbo Pascal Version 4.0
b. Compiler Version - 4.0
c. Target Hardware - IBM PC/AT or close compatible
d. Operating System - Microsoft MS-DOS (Version 3.x)
e. Program Description

Director is a set of functions and procedures that allow the output MS DOS file directories to a windowed environment. Masking options and a selector for normal or abbreviated (similar to the MS-DOS /w switch) displays are allowed.

2. Subroutines Contained

a. StandBy

(1) Type: Procedure

(2) Purpose: Used internally by ShowDir, this procedure displays an operator prompt to pause long listings. The procedure exits when a key is pressed.

(3) Description of Parameters:

Input: Operator input from System.ReadKey

Output: Prompt information to the window supplied by the calling program.

(4) Subroutines Called:

CRT.GoToXY
CRT.HighVideo
CRT.WhereX
CRT.WhereY
System.ReadKey

(5) Process Description

The procedure notes the position of the cursor, writes a prompt to the operator, and waits until the operator presses a key. The procedure then blanks the prompt, and exits.

b. View_Dir

(1) Type: Procedure

(2) Purpose: Provides the selective display of a directory, with the filenames and subdirectories displayed in summary form (no date, size or attribute data).

(3) Description of Parameters:

Input: MatchPtrn, which specifies the path and wildcard options; FromLine and ToLine, which specify the window size.

Output: To the window supplied by the calling program.

(4) Subroutines Called:
CRT.GoToXY
CRT.HighVideo
CRT.Lowvideo
DOS.FindFirst
DOS.FindNext

(5) Process Description
The procedure positions the cursor at column one of the line specified in FirstLine, then utilizes the procedure FindFirst to find any file or directory matching MatchPtrn. This sets up the DOS unit for subsequent searches. The first entry found is displayed and then FindNext is used for subsequent entries until the directory is exhausted. Subdirectories are displayed in highlighted video for ease of recognition in this summary display.

c. WriteEntry

(1) Type: Procedure

(2) Purpose: Displays the complete file or directory information of attributes, size, date and time for procedure ShowDir.

(3) Description of Parameters:
Input: DirInfo, a DOS Unit structure that contains packed information about the most recently found directory entry; line, the window line to display the information on. Output: To the window supplied by the calling program.

(4) Subroutines Called:
GetAttribut
CRT.GoToXY
CRT.HighVideo
CRT.Lowvideo
DOS.FindFirst
DOS.FindNext
DOS.UnPackTime

(5) Process Description
The procedure calls library procedures in the DOS unit to unpack the time entry in DirInfo. GetAttribut maps the attribute order to a string representation. The name, "<DIR>" designation or file size, creation date and time, and the attribute string are then written on the display at Line in MS-DOS format.

d. GetAttirbut

(1) Type: Procedure

(2) Purpose: Map an MS-DOS attribute number to a text string.

(3) Description of Parameters:
Input: attr, the ordinal MS-DOS attribute combination.

Output: attribut, a string to return the text string representation of the attribute.

(4) Subroutines Called:
System.Str

(5) Process Description

The attr variable is used as a selector in a case construct to load attribut with the proper string. If the variable does not map, the hexadecimal number in the variable attr is converted to a string for display.

e. Show_Dir

(1) Type: Procedure

(2) Purpose: Provides the selective display of a directory, with the filenames and subdirectories displayed in summary form (no date, size or attribute data).

(3) Description of Parameters:

Input: MatchPtrn, which specifies the path and wildcard options; FromLine and ToLine, which specify the window size; error, which reports DOSerror back to the calling program.

Output: To the window supplied by the calling program.

(4) Subroutines Called:

CRT.ClrEOL
CRT.ClrScr
CRT.GoToXY
CRT.HighVideo
CRT.Lovideo
Director.WriteEntry
DOS.FindFirst
DOS.FindNext
System.INC

(5) Process Description

The procedure utilizes the procedure FindFirst to find any file or directory matching MatchPtrn. This sets up the DOS unit for subsequent searches. Depending on the state of DOS.DOSError, which indicates error conditions on the attempt to find a directory entry, the entry is either displayed via WriteEntry or an error or status message is displayed and the procedure exits. The first entry found is displayed and then FindNext is used for subsequent entries until the directory is exhausted. For directories that exceed the window size specified by FromLine and ToLine, the display is paused by a call to the procedure StandBy and the operator is allowed to press a key to continue.

APPENDIX G

MAINTENANCE MANUAL FOR UNIT ERRORCOD

A. UNIT ERRORCOD

1. Configuration Information

a. **Language - Turbo Pascal Version 4.0**
b. **Compiler Version - 4.0**
c. **Target Hardware - IBM PC/AT or close compatible**
d. **Operating System - Microsoft MS-DOS (Version 3.x)**
e. **Program Description**

ErrorCod is a array of string constants mapped by the DOS Error Code, Error Class, Recommended Error Action and Error Locus indices found in (Microsoft, 1986, pp. 3-1 - 3.11, 4.254 - 4.255). The unit is used by the units Parser, Spawn and the program Distrib to report errors. A procedure is also provided to retrieve extended error code information available in MS-DOS versions 3.0 and above by DOS function call $59.

2. Subroutines Contained

a. **Extended_Error_Code**

(1) Type: Procedure

(2) Purpose: To return the extended error code, class and locus information available in MS DOS version 3.0 and later, in response to a DOSERROR result.

(3) Description of Parameters: Extended_Error_Code returns the Error Code, Error Class and Error Locus in the respective variables.

(4) Subroutines Called:
DOS.Intr($21)

(5) Process Description

This procedure calls DOS function $59 with register BX = 0 to get extended error information from MS DOS following an operating system error flag, as indicated by the Turbo Pascal variable DOSERROR > 0.

APPENDIX H

MAINTENANCE MANUAL FOR UNIT GENERAL

A. UNIT GENERAL

1. Configuration Information
   a. Language - Turbo Pascal Version 4.0
   b. Compiler Version - 4.0
   c. Target Hardware - IBM PC/AT or close compatible
   d. Operating System - Microsoft MS-DOS (Version 3.x)
   e. Program Description

General is a collection of general purpose routines that support the Wndow Unit and other modules.

2. Subroutines Contained

   a. FillWord
      (1) Type: Procedure
      (2) Purpose: Given a variable, V, the procedure fills Num words in the variable with integer Value.
      (3) Description of Parameters:
      Input: Variable V; Num, the number of words to be filled; and Value, the fill value.
      Output: V is returned after filling.
      (4) Subroutines Called:
      Inline assembly
      (5) Process Description
      Register DI is initialized with the starting offset of the variable V, CX contains the number of words to be filled, and AX contains the Value to be used to fill. The STOSW instruction autoincrements the DI register after each store and decrements CX. The loop ends when CX = 0. Using assembly language string processing instructions, the procedure uses the DI index register to point to the memory iterates a store operation with the 16 bit word Value beginning at the first location in V and continuing for Num iterations, incrementing the storage location by a 16 bit word each time.

   b. Exchange
      (1) Type: Procedure
      (2) Purpose: Exchange the contents of two variables without compatibility checking.
      (3) Description of Parameters:
      Input: S, D are the variables to be exhanged, and L is the number bytes to be exchanged.

Output: The variables S and D are returned after the exchange.

(4) Subroutines Called:
Inline assembly

(5) Process Description
Register DI is loaded with the offset of variable S, register SI with that of D. CX receives L. The value at variable D, indexed by DI, is loaded into AX and exchanged with the value at variable S, indexed by SI. STOSB autoincrements both index registers and decrements CX. The loop stops as CX reaches 0.

c. **Beep**

(1) Type: Procedure

(2) Purpose: Produce a speaker tone for 1/4 second.

(3) Description of Parameters:
Input: Freq, the desired tone frequency.
Output: A speaker tone.

(4) Subroutines Called:
CRT.Delay
CRT.Sound
CRT.NoSound

(5) Process Description
CRT procedures NoSound and Sound operate in tandem. First the speaker is silenced. Then, the Sound procedure in the CRT Unit is called with parameter Freq and a delay of 1/4 second is allowed before turning the speaker off again.

d. Max

(1) Type: Function

(2) Purpose: Returns the larger of two integers. Typically used with Open_Window to insure the window is large enough to hold a menu display.

(3) Description of Parameters:
Input: X, Y, the integers to be compared.
Output: The larger integer of the input parameters.

(4) Subroutines Called: None.

(5) Process Description
The two integers are compared and the function result equated to the larger.

e. Min

(1) Type: Function

(2) Purpose: Returns the smaller of two integers. Typically used with Open_Window to insure the window is large enough to hold a menu display.

(3) Description of Parameters:
Input: X, Y, the integers to be compared.
Output: The smaller integer of the input parameters.

(4) Subroutines Called: None.

(5) Process Description

The two integers are compared and the function result equated to the smaller.

f. Cursor_Size

(1) Type: Function

(2) Purpose: Sets the cursor displayed as either an underline or a block.

(3) Description of Parameters:

Input: Cursor_Type an enumerated type consisting of line, block or invisible. Mono is TRUE if the display is monochrome, FALSE if color.

Output: The video card is updated to display the selected cursor.

(4) Subroutines Called:

DOS.Intr($10) (video service)

(5) Process Description

Register AX is set to $10 to call the BIOS video service, and the CX register is set to the proper value for the cursor requested prior to the call.

g. Get_Time

(1) Type: Function

(2) Purpose: Returns a string with the current time.

(3) Description of Parameters:

Input: Nothing.

Output: A string with the current time in format HH:MM:SS xM.

(4) Subroutines Called:

DOS.Intr($21) (DOS service)

System.Str

(5) Process Description

Register AH is set to $2C to call the DOS time service, and the CH, CL, DH and DL return the ordinal number for hours, minutes, seconds and hundreths of seconds (Norton, 1985, p. 287). The Turbo Pascal Str procedure is used to convert each number into a string representation. The strings are then concatenated with formatting characters and AM or PM notations.

APPENDIX I

MAINTENANCE MANUAL FOR UNIT MISCPACK

A. UNIT Miscpack

1. Configuration Information

a. Language - Turbo Pascal Version 4.0
b. Compiler Version - 4.0
c. Target Hardware - IBM PC/AT or close compatible
d. Operating System - Microsoft MS-DOS (Version 3.x)
e. Program Description

Miscpack is a collection of data types and utility routines supporting these other units: Xmodm, Parser, Spawn, Redirect, and the main program Distrib. The strong typing features of Turbo Pascal require that instances data types in different units that must be equated be declared in one place to be compatible at compile time.

2. Subroutines Contained

a. BumpStrup

(1) Type: Procedure

(2) Purpose: To convert any string to upper case characters.

(3) Description of Parameters: S is the formal variable for a string of any length, since length checking is relaxed.

(4) Subroutines Called:
System.UpCase
System.Length

(5) Process Description
This procedure returns the string as a call by reference parameter after converting all of the characters making up the string to uppercase.

APPENDIX J

MAINTENANCE MANUAL FOR UNIT PARSER

A. UNIT PARSER

1. Configuration Information

a. Language - Turbo Pascal Version 4.0
b. Compiler Version - 4.0
c. Target Hardware - IBM PC/AT or close compatible
d. Operating System - Microsoft MS-DOS (Version 3.x)
e. Program Description

The central procedure in this unit is Parser_Main, which attempts to parse and execute an MS-DOS style command on the local machine. The remaining procedures and functions support this function.

2. Subroutines Contained

a. argc

(1) Type: Function

(2) Purpose: Returns the number of arguments in the command line parsed by the procedure Parse. Parse must be called before this function is valid.

(3) Description of Parameters:

Input: None.

Output: The number of arguments in the command line last parsed.

(4) Subroutines Called:

(5) Process Description

argc is set to the variable arg_count, which is loaded by Parse.

b. argv()

(1) Type: Function

(2) Purpose: Return the arg_count'th argument encountered on the last command line parsed by procedure Parse. Parse must be called before this function is valid.

(3) Description of Parameters:

Input: arg_count, the index of the argument desired, arg_array, the index to the arguments indexed, and arg_string, a copy of the command.

Output: A string, up to 128 characters long, containing the arg_count'th argument.

(4) Subroutines Called: None.

(5) Process Description

Following a call to procedure Parse, the data structure arg_array is loaded with the relative index of the start of each argument in the command line parsed, and the length of that argument. A length of zero at that index indicates no argument was found. To construct the arg_count'th argument, the command saved in arg_string is copied starting at the index saved in the index field in the arg_count'th record of array arg_array, for the length field in the same record.

c. Init_Parse

(1) Type: Procedure

(2) Purpose: To parse the input string for Parser_Main, and initialize the component strings for later use.

(3) Description of Parameters:

Input: Command_s, an input parameter for Parse_Main.

Output: Pathspec is set to argv(0), the remaining drive, node, and name strings are parsed.

(4) Subroutines Called:

Parse.argv(0)

Parse.ParseName

(5) Process Description

This procedure is local to Parse_Main, and is used any time the command string being parsed is first parsed, or after the command has been modified.

d. Parse

(1) Type: Procedure

(2) Purpose: Set up the argv and arc functions for a command line received.

(3) Description of Parameters:

Input: Command, a string variable containing the command to be parsed.

Output: arg_array and arg_count are private variables visible inside this unit.

(4) Subroutines Called:

System.Inc

System.Length

(5) Process Description

First, a copy of the command is retained outside this procedure in arg_string for later use by argv. Arg_array is then initialized to clear old parsing actions, and arg_count is initialized to zero to act as an index for arg_array. The cycle begins by skipping leading whitespace in the command. When the first non whitespace character is encountered, the index of the string is noted in the arg_count'th record of arg_array and non whitespace characters are skipped while incrementing the length field to determine the length of the argument. Upon reaching whitespace again. the next record in arg_array is selected and the cycle repeats until the end of the string

is reached. arg_array, arg_count and arg_string are retained in variables private to the unit for future ust.

e. ParseName

(1) Type: Procedure

(2) Purpose: Break a complete filename with path and drive into its component parts.

(3) Description of Parameters:

Input: inName is a composite drive, path and filename string.

Output: The component file name, extension, name and extension, path, drive and node (if any) in inName.

(4) Subroutines Called:

System.Copy
System.Delete
System.Length

(5) Process Description

The syntax for inName is: [Node::][Drive:][\]directory[\directory\]filespec[/Switch], similar to the MS-DOS command line syntax with the exception of the node designator, which was intended for use with commands intercepted by a background process. The procedure scans the command line backwards, looking for the delimiters established in the constants Path_or_drive and Node_or_drive. When such delimiters are found, the suceeding substring is copied into the appropriate output variable and the command is truncated to continue the scan until the first character is reached. The filename, if any, is then broken down similarly into its component name and extension (Swan, pp. 26 - 27).

f. Resolve_Command

(1) Type: Function

(2) Purpose: This procedure is passes the first argument in a command line and attempts to create a complete path specification and match the filename to a command normally handled internally by the DOS command processor or to an executable file in the specified directory. Relative directory citations are adjusted to a path from the root directory. Parser_Main sets up the component parts of the first argument via Parse_Name and places them in the variables immediately above this function.

(3) Description of Parameters:

Input: Argurement, the first parameter in the command line from Parser_Main.

Output: Argument, corrected to a complete path specification and filename extension. The function returns the type of file detected (batch file, com file, executable file, directory, pathstring or other file) as an enumerated type.

(4) Subroutines Called:

System.GetDir
DOS.FindFirst
DOS.FindNext

(5) Process Description

Resolve_command first determines the current directory with GetDir, and adjusts any relative directory path specification found in argument to a full path specification complete with drive and root directory, if needed. This is needed by the Exec function called by Parser_Main. If no file extension was parsed by Parse_Name, Resolve_Command attempts to find an executable file in the directory cited by the now complete path specification by finding a file with the same name and an "COM", "EXE", or "BAT" extension. They are searched for in reverse priority so that the Exec call will attempt to execute the filename with the highest rank, as Command.Com does (Mefford, 1988, p. 336) and the file type is identified. If the command did cite a filename with extension, the file type is identified. The file type is returned by the function for Parser_Main. If an executable file was not found, a check is made to see if a directory by that name exists, otherwise a general pathname type is returned.

g. Parser_Main

(1) Type: Procedure

(2) Purpose: This procedure parses a command received by the Slave and attempts to execute it.

(3) Description of Parameters:

Input: Command_s, the received command string.

Output: Response and Error_Msg are strings containing either the command output and error messages, respectively, or filenames containing the information. Restype and Errtype tell the calling program what Response and Error_Msg contain. Prompt is the local machine current directory for return to the Master via the calling program after the response is competed.

(4) Subroutines Called:

Parser.InitParse
Parser.Match_Command
Parser.Resolve_Command
Parser.Parse
Parser.ParseName
Parser.argc
Parser.argv()
Spawn.Match_Command
Spawn.Process_intrinsic_command
Spawn.Run_local
System.Length

(5) Process Description

On entry, command_s contains the complete command to be executed. Its component arguments are isolated by Init_Parse, and then a special case is checked to see if a simple drive change is requested (e.g., "C:"). If so, the internal DOS command "CD" is prefixed to the command and it is re-parsed. The filename in the first argument is checked by Spawn.Match_Command against a set of commands that this program handles internally. This is a subset of the MS-DOS internal commands: Change Directory, Copy, Delete, Directory, Erase,

Make Directory, Remove Directory, Rename and their abbreviated forms. If matched, the command is passed to Spawn.Process_Intrinisic_Command for execution and collection of responses. If not, the file type returned by Resolve_Command is used as a case selector to either run an executable file via Spawn.Run_Local, or a syntax error indication is returned to the calling program. If executable, the command (program name) is separated from the following command tail and passed to Run_Local.

APPENDIX K

MAINTENANCE MANUAL FOR UNIT REDIRECT

A. UNIT REDIRECT

1. Configuration Information

a. **Language - Turbo Pascal Version 4.0**
b. **Compiler Version - 4.0**
c. **Target Hardware - IBM PC/AT or close compatible**
d. **Operating System - Microsoft MS-DOS (Version 3.x)**
e. **Program Description**

Redirect is a set of functions and procedures that allow the output of programs spawned under the Slave computer's copy of the main program Distrib to be redirected to files. Once the program ends, the Slave computer can then forward the output normally displayed on the screen to the Master computer for display.

2. Subroutines Contained

a. **Init_Redirect_Unit**

(1) Type: Procedure

(2) Purpose: To reverse the Turbo Pascal initialization of the Pascal standard files Input and Output to the CRT Unit in preparation for redirection.

(3) Description of Parameters: None. This procedure reassigns the Pascal standard files Input and Output.

(4) Subroutines Called:
System.Assign
System.Reset
System.Rewrite

(5) Process Description

The Turbo Pascal Version 4.0 reference manual indicates that the initialization code found in standard Unit CRT assigns the Pascal standard test files Input and Output to the CRT Unit. In order to accomplish I/O redirection, these files must be rereferenced to the standard input and output. The above subroutines accomplish this.

b. **Duplicate_Handle**

(1) Type: Function

(2) Purpose: Returns a second handle that refers to the same file (or device) as the variable Handle. Used to save the reference to standard I/O for later restoration after redirection ends.

(3) Description of Parameters: Handle is the file handle to be duplicated. ErrorNum is a variable for an MS-DOS error code returned in the AX register if the MS-DOS function call fails.

(4) Subroutines Called:
DOS.Intr(#21)
(5) Process Description
The DOS.Intr(#21) call is to the Duplicate_Handle function, #45. The function returns another handle of type word.

c. Close_File_Handle

(1) Type: Function
(2) Purpose: Closes a file handle that refers to a file or device. Used to terminate I/O to the standard input or output handle when redirected, and to dispose of the redirection handle. ErrorNum is a variable for an MS-DOS error code returned in the AX register if the MS-DOS function call fails.
(3) Description of Parameters: Handle is the file handle to be closed.
(4) Subroutines Called:
DOS.Intr(#21)
(5) Process Description
The DOS.Intr(#21) call is to the Close_Handle function, #3E. ErrorNum is returned with an MS-DOS error code if the call fails, as indicated by a FALSE function result.

d. Redirect_Handle

(1) Type: Procedure
(2) Purpose: Forces a handle used by the system for standard input or output to be redirected to the same file or device as another handle. The file or device originally pointed to may then closed. I/O to the standard input or output handle now appears at the same file or device as the handle redirected to.
(3) Description of Parameters: Handle is the file handle pointig to the file or device to be redirected to, Red_Handle is the standard I/O handle to be redirected.
(4) Subroutines Called:
DOS.Intr(#21)
(5) Process Description
The DOS.Intr(#21) call is to the FDup_Handle function, #46. ErrorNum is returned with an MS-DOS error code if the call fails. On return the redirected standard I/O handle now operates through the file or device of Handle.

e. Redirect_Std_Output

(1) Type: Function
(2) Purpose: Redirects Standard Output to a file of our choosing.
(3) Description of Parameters: StdOut is the MS-DOS standard output file handle to be redirected. Std_Output_File_Temp is the file that output will be redirected to.
(4) Subroutines Called:
Redirect.Duplicate_Handle
Redirect.Redirect_Handle

(5) Process Description

The temporary output file is opened, a handle pointing to StdOut is saved and then StdOut is forced to point to our output file.

f. Restore_Std_Output

(1) Type: Function

(2) Purpose: Restores the saved standard Output to its previous state, sets a variable Response_File to the name of the file holding the redirected output to end redirection.

(3) Description of Parameters: StdOut is the MS-DOS standard output file handle that was redirected. Std_Output_File_Temp is the file that output was redirected to. Saved_Std_Out is the handle that points to the original standard Output.

(4) Subroutines Called:

Redirect.Close_File_Handle

Redirect.Redirect_Handle

(5) Process Description

StdOut, the file handle for standard output is reset to point to Saved_Std_Out, the temporary file Std_Output_File is closed for writing, and the variable Response_File is set to the name of the temporary file if no errors are encountered, otherwise NIL.

g. Redirect_Std_Input

(1) Type: Function

(2) Purpose: Redirects standard Input to be drawn from a file of our choosing.

(3) Description of Parameters: StdIn is the MS-DOS standard input file handle to be redirected. Std_Input_File_Temp is the file that input will be redirected from.

(4) Subroutines Called:

Redirect.Duplicate_Handle

Redirect.Redirect_Handle

(5) Process Description

The temporary input file is opened for reading, a copy of the handle pointing to StdIn is saved and then StdIn is forced to point to our input file.

h. Restore_Std_Input

(1) Type: Function

(2) Purpose: Restores the saved standard Input to its previous handle, and closes the input file to end redirection.

(3) Description of Parameters: StdIn is the MS-DOS standard input file handle that was redirected. Std_Input_File_Temp is the file that input was redirected from. Saved_Std_In is the handle that points to the original standard Input.

(4) Subroutines Called:

Redirect.Close_File_Handle

Redirect.Redirect_Handle

(5) Process Description

StdIn, the file handle for standard input is reset to point to Saved_Std_In, the temporary file Std_Input_File is closed for reading. The function returns TRUE if no file errors are detected.

i. Redirect_Std_Error

(1) Type: Function

(2) Purpose: Redirects standard Error to be sent to a file of our choosing.

(3) Description of Parameters: StdErr is the MS-DOS standard error file handle to be redirected. Std_Error_File_Temp is the file that error will be redirected to.

(4) Subroutines Called:

Redirect.Duplicate_Handle

Redirect.Redirect_Handle

(5) Process Description

The temporary error file is opened for writing, a copy of the handle pointing to StdErr is saved and then StdErr is forced to point to our error file.

j. Restore_Std_Error

(1) Type: Function

(2) Purpose: Restores the saved standard Error to its previous handle, and closes the error file to end redirection.

(3) Description of Parameters: StdErr is the MS-DOS standard error file handle that was redirected. Std_Error_File_Temp is the file that Error was redirected to. Saved_Std_Error is the handle that points to the original standard Error.

(4) Subroutines Called:

Redirect.Close_File_Handle

Redirect.Redirect_Handle

(5) Process Description

StdErr, the file handle for standard error is reset to point to Saved_Std_Error, the temporary file Std_Error_File is closed for reading. The function returns TRUE if no file errors are detected.

k. Redirect_All_Output

(1) Type: Function

(2) Purpose: Redirects both standard error and standard output to a file of our choosing.

(3) Description of Parameters: StdOut is the MS-DOS standard output file handle to be redirected. Std_Output_File_Temp is the file that output will be redirected to. StdErr is the MS-DOS standard errort file handle to be redirected. Std_Error_File_Temp is the file that output will be redirected to.

(4) Subroutines Called:

Redirect.Duplicate_Handle

Redirect.Redirect_Handle

(5) Process Description

The temporary output file is opened, a handle pointing to StdOut is saved and then StdOut is forced to point to our

output file. The process is repeated for StdErr, except that it is redirected to the same output file.

l. Restore_All_Output

(1) Type: Function

(2) Purpose: Restores the saved standard output and error to their previous states, sets a variable Response_File to the name of the file holding the redirected output to end redirection.

(3) Description of Parameters: StdOut is the MS-DOS standard output file handle that was redirected. Std_Output_File_Temp is the file that output was redirected to. Saved_Std_Out is the handle that points to the original standard Output. StdErr is the MS-DOS standard output file handle that was redirected. Std_Error_File_Temp is the file that output was redirected to. Saved_Std_Err is the handle that points to the original standard Error.

(4) Subroutines Called:

Redirect.Close_File_Handle
Redirect.Redirect_Handle

(5) Process Description

StdOut, the file handle for standard output is reset to point to Saved_Std_Out, the temporary file Std_Output_File is closed for writing. StdErr, the file handle for standard error is reset to point to Saved_Std_Err, the temporary file Std_Error_File is closed for writing, and the variable Response_File is set to the name of the temporary file if no errors are encountered, otherwise NIL.

m. Restore_CRT_Assignments

(1) Type: Procedure

(2) Purpose: To set the standard Input and Output files to textdrivers in the CRT Unit. Faster inoput and output is obtained.

(3) Description of Parameters: None. This procedure reassigns the Pascal standard files Input and Output to CRT.AssignCRT (Input) and CRT.AssignCRT (Output).

(4) Subroutines Called:

System.AssignCRT
System.Reset
System.Rewrite

(5) Process Description

The assignments restore the input and output standard files to the CRT unit.

APPENDIX L

MAINTENANCE MANUAL FOR UNIT SPAWN

A. UNIT SPAWN

1. Configuration Information
   a. Language - Turbo Pascal Version 4.0
   b. Compiler Version - 4.0
   c. Target Hardware - IBM PC/AT or close compatible
   d. Operating System - Microsoft MS-DOS (Version 3.x)
   e. Program Description

This unit detects commands that should be processed internally by the Distrib program, and executes commands internally or by spawning a child process. Command output and error responses are returned to the caller either as strings suitable for conversion to Xmodm buffers, or by reference to files containing the text. This unit also contains the redirection switch as a public variable that dictates whether program output will be redirected to a file or displayed locally on the screen. This switch is normally set to redirect to file.

2. Subroutines Contained

   a. Match_Command
      (1) Type: Function
      (2) Purpose: To search a command string for a substring that matches a command to be processed internally by the Slave program.
      (3) Description of Parameters:
      Input: Filespec is a command stripped of path specification, or leading or trailing spaces.
      Output: The function returns TRUE if a match was found, along with an enumerated type matching the command, FALSE otherwise.
      (4) Subroutines Called:
      System.Length
      System.Pos
      (5) Process Description
      A substring search is conducted using the enumerated internal command type to index an array of strings containing the command names. The internal command must be matched by exact replication and must be positioned as the first substring in FileSpec.

   b. Process_Intrinsic_Command
      (1) Type: Procedure

(2) Purpose: This procedure executes an internal command detected by Match_Command. This procedure, and Run_local, execute commands for Spawn.Parser_Main.

(3) Description of Parameters:

Input: Command, the enumerated type specifying the internal command. Command_tail are the parameters for the internal command.

Output: Response and Error_Msg are strings containing either the command output and error messages, respectively, or filenames containing the information. Restype and Errtype tell the calling program what Response and Error_Msg contain. Prompt is the local machine current directory for return to the Master via the calling program after the response is competed.

(4) Subroutines Called:

System.ChDir
System.GetDir
System.Mkdir
System.RmDir

(5) Process Description

The Command parameter is used in a CASE construct select commands that are completed by Turbo Pascal functions and procedures, and to pass other internal commands to Run_local to spawn a copy of the MS-DOS command processor and run the command. This approach is taken to greatly simplify the command parsing and execution, since these requirements can be offloaded to the spawned command processor for commands with complex processing requirements such as DIR. Batch_mode is set to signal Run_Local to spawn a copy of the command processor rather than attempting to execute the command as a program.

c. **Run_Local**

(1) Type: Procedure

(2) Purpose: This procedure executes all command that nd detected by Match_Command. This procedure, and Process_Intrisic_Command, execute commands for Spawn.Parser_Main.

(3) Description of Parameters:

Input: Program_name, the name of the command or file to be executed; Command_line, the arguments for the command or file; and Batch, which signals that a copy of the MS-DOS command processor is to be used to run the program for batch files and certain internal MS-DOS commands.

Output: Response and Error_Msg are strings containing either the command output and error messages, respectively, or filenames containing the information. Restype and Errtype tell the calling program what Response and Error_Msg contain. Prompt is the local machine current directory for return to the Master via the calling program after the response is competed.

(4) Subroutines Called:

Redirection.Init_Redirection_Unit
Redirection.Redirect_All_Output
Redirection.Restore_All_Output

Redirection.Restore_CRT_Assignments
Support.Find_Environment
System.ChDir
System.GetDir
System.UpCase
System.Length

(5) Process Description

CRT.CheckBreak is set to allow an operator to terminate execution of a runaway program. If the Batch flag is set, the command is adjusted to execute a copy of COMMAND.COM and the original command and arguments are moved to command tail. Find_Environment is used to locate the explicit path specification and file name for COMMAND.COM, as required by the Exec procedure. The current directory is saved to return the program to its working directory after command execution. If the Redirection flag has been set, calls are made to the Redirection Unit to route all subsequent program output to files visible in the Redirection Unit. This redirection is inherited by any programs spawned from this program by Exec (Greco, 1987, p. 25). Exec is then called to spawn the program(s). On return, the standard output handles are restored and the original working directory restored as a precaution.

APPENDIX M

MAINTENANCE MANUAL FOR UNIT SUPPORT

A. UNIT SUPPORT

1. Configuration Information

a. Language - Turbo Pascal Version 4.0
b. Compiler Version - 4.0
c. Target Hardware - IBM PC/AT or close compatible
d. Operating System - Microsoft MS-DOS (Version 3.x)
e. Program Description

The Support Unit contains most of the constant declarations for the program, along with the initialization procedure some general purpose procedures. From (Edwards, 1987, pp. 241 - 272).

2. Subroutines Contained

a. Initialize

(1) Type: Procedure

(2) Purpose: This procedure sets the default parameters for the program, attempts to read the telephone number file and creates a file if none exists, reads the user developed configuration file to override some defaults, displays the terminal screen and initializes the Wndow Unit.

(3) Description of Parameters:
Input:
Output:

(4) Subroutines Called:
System.UpCase
System.Length

(5) Process Description

This procedure first attempts to open a configuration file under the name found in the constant structure Defaults. If this file exists, the current configuration is read in to a similar structure called Current, otherwise all parameters are taken from the constant structure. This is used to set the screen colors, identify the initial communications port to use, and identify the modem port. This file may be updated from the Master screen. From the configuration selected, the environmental parameters are established. A similar process attempts to read the list of telephone numbers and associated parameters, however the size of this array is not known in advance. A memory block is drawn from the heap for each telephone record read to make the list. If the file does not exist, a dummy record is established. This file may also be updated from the screen. Finally, the designated communications port is initialized. This is

essential if the Slave computer is to recognize external commands without operator intervention.

b. **Save_File**

(1) Type: Procedure

(2) Purpose: To save user modified configuration or telephone dialing list parameters in a local file for later use on program initialization.

(3) Description of Parameters: D is a boolean switch that selects the file to be saved.

(4) Subroutines Called:

Wndow.Open_Window
CRT.ClrScr
Support.Yes
Support.NoFile
Support.OK
Wndow.Close_Window

(5) Process Description

This procedure saves the default environmental parameters as modified by the user in the file DISTRIB.CFG; or the current list of telephone numbers and communications port parameters in the file DISTRIB.PHN. Both files are loaded on program initialization (if available) and override the default parameters found in the constant data structures in the unit Support.

c. OK

(1) Type: Procedure

(2) Purpose: To obtain an acknowledgement from the user.

(3) Description of Parameters:

Input: S, the string to title the prompt window.
Output: The user has responded if the call returns.

(4) Subroutines Called:

Wndow.Open_Window
Wndow.Process_Window
Wndow.Close_Window

(5) Process Description

This function opens a window with a "OK" display and the query in the window title field. The operator then depresses the ENTER key to acknowledge, which is detected by Process_Window. The widow is closed and the procedure call returns.

d. Yes

(1) Type: Function

(2) Purpose: To prompt the user for a yes or no response.

(3) Description of Parameters:

Input: S, the string to title the prompt window.
Output: The function returns true if Yes was selected.

(4) Subroutines Called:
Wndow.Open_Window
Wndow.Process_Window
Wndow.Close_Window

(5) Process Description

This function opens a window with menu bar, displaying the query in the window title field and the selections "Yes" or "NO" in the window. The operator selects with the menu bar, and Process_Window returns a value of two if the selection was "Yes." The widow is closed and the function returns true if "Yes" was selected.

e. NoFile

(1) Type: Procedure

(2) Purpose: To obtain an acknowledgement from the user after failing to find a file.

(3) Description of Parameters:

Input: S, the string to title the prompt window.

Output: The user has responded if the call returns.

(4) Subroutines Called:
CRT.ClrScr
Support.OK
Wndow.Open_Window
Wndow.Process_Window
Wndow.Close_Window

(5) Process Description

This function opens a window to inform the operator that the desired file could not be found, then opens another window with a "OK" display. The operator then depresses the ENTER key to acknowledge, which is detected by the OK procedure. The widow is closed and the procedure call returns.

f. Build_Status_Line

(1) Type: Procedure

(2) Purpose: To construct a status line at the bottom of the video display.

(3) Description of Parameters:

Input: Nothing.

Output: A status line containing information on the current communications port is displayed at the bottom of the screen.

(4) Subroutines Called:
System.Insert
Wndow.Write_Status_Line

(5) Process Description

The procedure starts with a blank status line and inserts substrings depending on the state of variables declared in this unit to construct the status line. Write_Status_Line displays the line in the appropriate position.

g. Check_keyboard

(1) Type: Function

(2) Purpose: To return a keyboard character, including special characters.

(3) Description of Parameters:

Input: The key is taken from the Readkey function. Output: The function returns the character read, or the keyboard scan code in the high byte if a special character is read (Readkey returned a zero). If no key is available, the function returns zero.

(4) Subroutines Called:

System.KeyPressed
System.Readkey

(5) Process Description

The function checks the Keypressed function and if true, calls Readkey to get the character. If Readkey returns zero, a special key has been pressed, and the scan code is read from Readkey. The character is returned, or the scan code in the high byte of the integer if appplicable.

h. Check_Auxport

(1) Type: Function

(2) Purpose: This function checks for a character at the currently selected communications port and returns a result.

(3) Description of Parameters:

Input: Nothing.

Output: NUL if no character is ready, or the character if one was read.

(4) Subroutines Called:

DataCom.RS232_Avail
DataCom.RS232_In

(5) Process Description

RES232_Avail returns true if a character is available in the receive buffer of the currently selected communications port. If true, the character is read through RS232_In, and passed to the LST device and Ascii_file if public variables are set. The character is returned, or NUL if no character was available.

i. Find_Environment

(1) Type: Function

(2) Purpose: To return a specified string from the operating system environment. This function typically is called to find the COMSPEC=<path specification> string to locate a copy of the MS-DOS command processor. With this path information, a second copy of the command processor can be spawned to run programs from this one.

(3) Description of Parameters:

Input: What is the parameter to be searched for. The environment contains strings of the form What=<text>.

Output: If found, the <text> part of the environment string; if not, a NUL string.

(4) Subroutines Called:
System.MemW
System.Ptr
System.Copy
System.Length

(5) Process Description

To run a batch file, a second copy of the MS-DOS command processor is spawned as a child process, with the batch file as a command tail. The secondary processor executes the batch file and terminates. A copy of the command processor must first be located without previous knowledge. MS-DOS normally places a string citing the path to the COMMAND.COM on system initialization in an area of memory called the environment, along with other information from the AUTOEXEC.BAT file such as PATH information. A segment pointer to this MS-DOS environment is placed in any program spawned from the original command processorin the child Program Segment Prefix, at offset $002C. The environment starts on a segment boundary, so the offsed is automatically $0. This environment is the same one manipulated by the SET command from MS-DOS, and normally contains a string of the form COMSPEC=D:\directory\directory\command.com. To search the environment for the requested string, a pointer (Environ) is typed for the maximum size of the environment, 32K bytes and initialized from the segment value at offset $002C. Each string in the environment is terminated by a NUL character (ASCIIZ). The environment area itself is terminated by an extra NUL. The environment area is searched, string by string by copying the strings into a local variable string, S. Each of these strings is examined for the search string What. If found, the remainder of the string is returned, otherwise a NUL string. This function is duplicated in Unit Support to prevent circular unit dependencies. (Edwards, 1987, p. 250).

j. **Update_Status**

(1) Type: Procedure

(2) Purpose: To display or refresh the current status of the calling program in a monitor window.

(3) Description of Parameters:

Input: Typically this procedure writes current information contained in a data structure by writing formatted strings to an open window, and then displaying the contents of the data as a string, or by mapping an enumerated data type to an array of constant strings to display the value.

Output: A window display of the current status.

(4) Subroutines Called:
Wndow.Get_Window
CRT.ClrEOL
CRT.GoToXY

(5) Process Description

This procedure is local to Modify_Entry. The process depends on the caller to open a properly sized window and to set a variable called Status_ID to allow the status window to be accessed via Get_Window. Once reopened, the procedure writes the

current status information.The procedure then resets the working window to that of the caller's Monitor_ID.

k. **Modify_Entry**

(1) Type: Procedure

(2) Purpose: to display the current list of telephone numbers that may be dialed automatically, or the current program configuration parameters.

(3) Description of Parameters:

Input: I, a selector. If I > 0 the phone list is to be modified, if I = 0 then the configuration parameters are modified.

Output: The user is offered the opportunity to save the modifications to a file.

(4) Subroutines Called:

Update_Status (local)
CRT.ClrScr
CRT.GoToXY
System.UpCase
System.Length
Wndow.Open_Window
Wndow.Process_Window
Wndow.Close_Window

(5) Process Description

Depending on I, the procedure opens a window of the correct size, and then displays the current parameters by mapping their values through arrays of constant strings to display readable values. The procedure then enters a loop for operator entry of parameters to be modified. The user then positions a menu bar over the appropriate selection and presses ENTER. Depending on the selection, the procedure prompts the operator for an input string, or displays another parameterized window and calls Process_Window to obtain the current selection. When ESC is pressed, the loop ends and the recorded modifications may be safec to a configuration or phone list file by Save_File. All windows are closed and the procedure returns.

APPENDIX N

MAINTENANCE MANUAL FOR UNIT WNDOW

A. UNIT WNDOW

1. Configuration Information

a. Language - Turbo Pascal Version 4.0
b. Compiler Version - 4.0
c. Target Hardware - IBM PC/AT or close compatible
d. Operating System - Microsoft MS-DOS (Version 3.x)
e. Program Description

This unit provides all window creation, memory allocation, display, menu bar processing, closure and memory deallocation functions for the program Distrib. The unit was changed from an include file to a unit, but not otherwise changed from that originally developed by the author in (Edwards, 1987, pp. 50-98). The purpose descriptions are from the author.

2. Subroutines Contained

a. SetColor
(1) Type: Procedure
(2) Purpose: Set the EGA foreground color for text display.
(3) Description of Parameters:
Input: Color, the code to set the color to.
Output: All future text will be displayed in the color selected.
(4) Subroutines Called:
CRT.TextColor
(5) Process Description
The color selected is stored in the variable Foreground, and a call is made to TextColor to set the screen foreground color in accordance with the EGA monitor standards.

b. SetBackGround
(1) Type: Procedure
(2) Purpose: Set the EGA background color for text display.
(3) Description of Parameters:
Input: Color, the code to set the color to.
Output: All future text will be displayed on a background of the color selected.
(4) Subroutines Called:
CRT.TextBackGround

(5) Process Description

The color selected is stored in the variable Background, and a call is made to TextBackGround to set the screen background color in accordance with the EGA monitor standards.

c. **Get_Dummy_Screen**

(1) Type: Procedure

(2) Purpose: Force the Screen variable to point to a dummy area on the heap.

(3) Description of Parameters:

Input: Screen, Screen_New (Public variables in this unit.

Output: Screen and Screen_New

(4) Subroutines Called: None.

(5) Process Description

Screen is initialized to point to the the start of the display area for the color or monochrome monitor in Init_Window_Info. This procedure saves this pointer in Screen_New and then fills Screen with the same information.

d. **Get_Real_Screen**

(1) Type: Procedure

(2) Purpose: To undo the work of Get_Dummy_Screen

(3) Description of Parameters:

Input: Screen, Screen_New (Public variables in this unit.

Output: Screen and Screen_New

(4) Subroutines Called: None.

(5) Process Description

Screen is initialized to point to the the start of the display area for the color or monochrome monitor in Init_Window_Info. Get_Dummy_Screen redirects the pointer Screen to a dummy area on the heap. This procedure restores Screen to its original setting.

e. **Build_Borders**

(1) Type: Procedure

(2) Purpose: Build a border of single or double lines around a window.

(3) Description of Parameters:

Input: Lines, specifying a single or double border. Active_Window, a public pointer in this unit to a window control block containing information about the size and current position of the window to be bordered.

Output: The output is a border written to the display to outline the window.

(4) Subroutines Called:

General.FillWord

System.Length

(5) Process Description

This procedure determines the window limits contained in the window control block pointed to by Active_Window, and places standard symbols in screen memory to outline the window.

f. Open_Window

(1) Type: Function

(2) Purpose: Open a window on the screen and draw a border around it.

(3) Description of Parameters:

Input: X1, Y1, X2, Y2 are the window coordinates; Flag is a bit mask of allowed functions for this window (borders, GOTO allowed within the window, relocatable and can be closed from the main program); Name is the window title to be displayed.

Output: 0 - window opened successfully; 1 - Invalid window coordinates; 2 - not enough memory (failure).

(4) Subroutines Called:

System.GetMem
System.MemAvail
System.Move
Wndow.Build_Borders

(5) Process Description

After checking the input parameters for valid coordinates and sufficient memory, the memory required to save the portion of the screen displayed by the window is allocated from the heap and the window is drawn with the appropriate colors and borders. Active_Window is advanced to this new window after adding it to the linked list of open windows.

g. Close_Window

(1) Type: Function

(2) Purpose: To close the window pointed to by Active_Window.

(3) Description of Parameters:

Input: Active_Window is a public pointer managed by this unit, and refers to the currently open window.

Output: The window is closed, and Active_Window is redirected to the previous window in the linked list of open windows. The function returns FALSE if successful, TRUE if an attempt was made to close a window with Active_Window^=NIL (no more windows open).

(4) Subroutines Called:

System.FreeMem
System.Move
Wndow.Build_Borders
Wndow.SetBackground
Wndow.SetColor

(5) Process Description

After checking the input parameters for valid coordinates and sufficient memory, the memory required to save the portion of the screen displayed by the window is allocated from the

heap and the window is drawn with the appropriate colors and borders. Active_Window is advanced to this new window after adding it to the linked list of open windows.

h. Save_Window

(1) Type: Function

(2) Purpose: This function saves the image of the current window, closes it, and returns a pointer to the saved window in memory.

(3) Description of Parameters:

Input: Active_Window is a public pointer managed by this unit, and refers to the currently open window.

Output: A pointer to the saved window.

(4) Subroutines Called:

Wndow.Open_Window

Wndow.Close_Window

(5) Process Description

W, a local variable is pointed to the same window_block as the current Active_Window. The procedure then opens a window with parameters identical to the current window by using the local pointer W to dereference the current window parameters. The act of opening a window of the same size and parameters has the effect of saving the original window. Active_Window now points to the new window. If the call to Open_Window fails, a NIL pointer is returned from Save_Window and the function exits. Otherwise, parameters from the saved window are transferred to the Active_Window block, W is redirected to the newly updated current window, Active_Window is retracted to the saved window and the window that overlaid it is closed. The function returns the pointer to the saved block.

i. Restore_Window

(1) Type: Procedure

(2) Purpose:

(3) Description of Parameters:

Input: A pointer to a saved window.

Output: TRUE if the function was unable to restore the window.

(4) Subroutines Called:

Wndow.Open_Window

Wndow.SetBackGround

Wndow.SetColor

(5) Process Description

The function first uses the window pointer to set the video display colors. Then, an attempt is made to open a window of the same size as the saved window. If this fails, the function returns true. Otherwise, the Active_Window parameters are set to the saved window, the saved window is added to the window control block chain, and Active_Window is reset to point to the restored window.

j. Get_Window

(1) Type: Function

(2) Purpose: To bring a window to the top of the screen.

(3) Description of Parameters:

Input: Which, the ID of the window to be surfaced.

Output: False if the operation succeeds, True if the ID did not exist.

(4) Subroutines Called:

Wndow.Get_Dummy_Screen
Wndow_Restore_Window

(5) Process Description

Get_Window follows the backlinks from Active_Window back until the ID of Which is found or the links end at a NIL. If found, Move_Window is used to copy the desired window into a heap area obtained by Get_Dummy_Screen. The window is then placed on the screen by Restore_Window.

k. **Move_Window**

(1) Type: Function

(2) Purpose: To move a current window by a relative X and Y.

(3) Description of Parameters:

Input: X, Y the direction and amount to move the window.

Output: False if the operation succeeds, True if the coordinates are invalid.

(4) Subroutines Called:

CRT.Window
Wndow.Exchange

(5) Process Description

Move_Window checks the values of X and Y and then copies the window incrementally in the desired direction(s). The built in procedure Window is then used to enable the new window location for display.

l. **Write_Status**

(1) Type: Procedure

(2) Purpose: To display a string on the 25th video display line with a video attribute.

(3) Description of Parameters:

Input: S, the status string; Attrib, the display attribute.

Output: The string is written to the display.

(4) Subroutines Called:

System.Length

(5) Process Description

The procedure first concatenates the attribute byte with the display character and then writes the combination to the screen as a word, using the Screen pointer.

m. **Process_Window_Menu**

(1) Type: Procedure

(2) Purpose: to display and process a menu in the current window.

(3) Description of Parameters:

Input: Menu is a constant that must consist of an integer, followed by an array of string constants of length Menu.

Output: The function returns a byte reflecting the index of the i'th string in the constant array. A zero is returned if ESC is pressed.

(4) Subroutines Called:

Set_Highlights (local)
GoDown (local)
GoHome (local)
GoEnd (local)
GoUp (local)
CRT.GoToXY
CRT.TextBackground
CRT.TextColor
Support.Max
Support.Min
System.Length
Wndow.Build_Borders

(5) Process Description

This function relies on a side effect of the data structure, and assumes that the array of strings representing the selections to be displayed in the window immediately follow Menu. By obtaining a memory address for Menu, the function opens a window of the proper size and then uses this implementation specific information to display the strings. The function then offers the operator the menu bar movement options on the status line to make a selection.

APPENDIX O

MAINTENANCE MANUAL FOR UNIT XMODM

A. UNIT XMODM

1. Configuration Information

a. **Language - Turbo Pascal Version 4.0**
b. **Compiler Version - 4.0**
c. **Target Hardware - IBM PC/AT or close compatible**
d. **Operating System - Microsoft MS-DOS (Version 3.x)**
e. **Program Description**

This unit handles all requests for Xmodem protocol packet and file transmission and reception.

2. Subroutines Contained

a. **String_to_Buf**

(1) Type: Procedure

(2) Purpose: Convert a string of length 128 to an Xmodem buffer of the same length.

(3) Description of Parameters:

Input: S, a 128 character string.

Output: buf, an Xmodem buffer. Short strings are padded with NUL characters.

(4) Subroutines Called:

System.Length

(5) Process Description

The string is treated as an array of characters, and each is read into the same position in the buffer.

b. **Buf_to_String**

(1) Type: Function

(2) Purpose: Convert a 128 character buffer into a string of the same length. Nonprinting characters are replaced with spaces.

(3) Description of Parameters:

Input: buf, the 128 character buffer of characters.

Output: s, a 128 character string.

(4) Subroutines Called: None.

(5) Process Description

The string is treated as an array of characters, and each character in the buffer, another array of compatible type is read into the string. Spaces are substituted for nonprinting characters.

c. **ReadAux**

(1) Type: Function

(2) Purpose: Returns a character from the currently selected communications port, and also writes the character to the monitor file and monitor window if selected. Provides a timeout function and a keypressed abort.

(3) Description of Parameters:

Input: Seconds, the number of seconds to wait for a character before returning with a timeout indication.

Output: A word with the received character in the low order byte, value 256 (timeout) otherwise.

(4) Subroutines Called:

CRT.Delay
CRT.Keypressed
CRT.TextColor
CRT.BackGround
DataCom.RS232_Avail
System.DEC

(5) Process Description

A factor is multiplied by the number of seconds to wait, and then used in a fast loop to test for a received character or operator keypress. Either event breaks the loop. If a character is available, the function returns the character. If Monitor_ID is greater than zero, a monitor window is open and the character is written to the cursor position there and to a monitor file. Otherwise, a timeout indicator is returned.

d. **WriteAux**

(1) Type: Procedure

(2) Purpose: Sends a character to the currently selected communications port, and also writes the character to the monitor file and monitor window if selected.

(3) Description of Parameters:

Input: Ch, the character to be sent.

Output: The character is sent and displayed if the Monitor_ID switch is greater than 0.

(4) Subroutines Called:

CRT.TextColor
CRT.BackGround
DataCom.RS232_Out

(5) Process Description

The character is sent out the communications port by RS232_Out. If Monitor_ID is greater than zero, a monitor window is open and the character is written to the cursor position there and to a monitor file.

e. **Send_String**

(1) Type: Procedure

(2) Purpose: To send a string out the currently selected communications port.

(3) Description of Parameters:

Input: S, a string.

Output: The string is sent to the port.

(4) Subroutines Called:

DataCom.RS232_Out

System.Length

(5) Process Description

The string is passed, character by character, to the communications port.

f. Receive_Record

(1) Type: Function

(2) Purpose: Receive an Xmodem packet from the currently selected communications port. A building block for file and command transfers.

(3) Description of Parameters:

Input: Buf, the data portion of the packet; Blocksize, the size of the data buffer; seconds, the number of seconds to wait before timing out on reception; and expected_block, the ordinal number of the next block expected from the sender.

Output: Buf is filled with the data packet contents if successfully received; errors indicates the number of errors encountered in receiving the packet.

(4) Subroutines Called:

Xmodm.ReadAux

Xmodm.WriteAux

(5) Process Description

Receive_Record first listens for the SOH character signalling the start of an Xmodm packet from the port via ReadAux, passing the numbert of seconds to wait on the call. The function exits immediately with an appropriate status code if a CAN, EOT or unexpected character is received. IF SOH is received, the function then assembles the Xmodem header, calculates a running checksum on the incoming data, and detects the checksum character. It then checks the packet for match between the block number and its inverse (packet locations two and three, respectively), an incorrect block number compared to the input expected_block, and a different checksum from that received and provides the appropriate status on return for each. If the packet was received correctly, an ACK is sent to the transmitter. If not, a NAK is sent.

g. Get_Buffer

(1) Type: Procedure

(2) Purpose: Reads a buffer of size blocksize from a previously opened file. Pads the buffer with NUL characters if smaller than requested.

(3) Description of Parameters:

Input: Buf, the buffer to fill; blocksize, the size of the buffer in bytes; XferFile is a private file variable in this unit.

Output: Buf contains the next file buffer.

(4) Subroutines Called:

System.BlockRead

(5) Process Description

The low level file read procedure BlockRead is used to read an untyped buffer. The procedure reports the number of bytes read. If less than the buffer size, the remaining bytes are filled with NULL characters.

h. Send_Record

(1) Type: Function

(2) Purpose: Send an Xmodem packet out the currently selected communications port. A building block for file and command transfers.

(3) Description of Parameters:

Input: Buf, the data portion of the packet; Blocksize, the size of the data buffer; seconds, the number of seconds to wait before timing out on acknowledgement; Block, the ordinal number of this packet; and errors, a count of the number of errors on the attempt to return to the calling program.

Output: Buf is unchanged and is a VAR parameter for efficiency; errors indicates the number of tries to send the packet.

(4) Subroutines Called:

DataCom.Purgeline
Xmodm.ReadAux
Xmodm.WriteAux

(5) Process Description

Send_Record first calculates a checksum value for the data in the buffer and then sends the SOH character signalling the start of an Xmodm packet to the port via WriteAux, followed by the block number and its inverse, the data and the calculated checksum value. PurgeLine is called to clear the receive buffer to prevent an erroneous interpretation of an earlier character received. ReadAux is then called to listen for the receiver's acknowledgement. Status is set accordingly. Finally, the keypressed function is checked to an operator interrupt and status is updated. Status is returned as the function result.

i. Sync_Receive

(1) Type: Function

(2) Purpose: Used to synchronize to receive Xmodem packets.

(3) Description of Parameters:

Input: Seconds, the number of seconds to wait between sending sync characters (NAK for Xmodem); and sync_character, the sync character to send.

Output: A status code indicating synchronization, timeout or operator keypress.

(4) Subroutines Called:

CRT.KeyPressed

DataCom.PurgeLine
DataCom.RS232_Avail
Xmodm.WriteAux

(5) Process Description

Sync_Recieve calculates the number of ten second intervals in seconds is calculated. The receive line is cleared and the sync character is sent. The function then loops waiting for a character to be received or the operator to press a key for the time indicated by seconds, sending a new sync character every five seconds. The function does not check the received character, only whether or not one was received in the allotted time. A status code is returned as the function result (packet acknowledged, negative acknowledge, receiver requests to cancel the transaction, timeout or operator keypress).

j. **Sync_Send**

(1) Type: Function

(2) Purpose: Used to synchronize to send Xmodem packets.

(3) Description of Parameters:

Input: Seconds, the number of seconds to wait between sending sync characters (NAK for Xmodem).

Output: A status code indicating synchronization, timeout or operator keypress.

(4) Subroutines Called:

CRT.KeyPressed
DataCom.PurgeLine
Xmodm.ReadAux

(5) Process Description

Sync_Send clears the receive line with PurgeLine and then calls ReadAux to detect a received character. A status code is returned as the function result (sync character received, checksum sync received, receiver timed out or a keypress was detected).

k. **Send_EOT**

(1) Type: Procedure

(2) Purpose: To signal the end of a data transfer for the Xmodem protocol.

(3) Description of Parameters:

Input: Status, to be changed to reflect the outcome of the call; and Suppress_EOT, a flag set to suppress the normal EOT on an Xmodem data transfer. Used to concatenate file transfers.

Output: Status, reflecting transmission completed, or a timeout error (or too many errors).

(4) Subroutines Called:

Xmodm.ReadAux
Xmodm.WriteAux

(5) Process Description

Suppress_EOT is first checked to see if the EOT will be sent. If TRUE, the EOT is not sent and the procedure returns a

completion status. This allows successive Xmodem transfers without encountering the normal flow control reversal. Otherwise, EOT characters are sent every ten seconds until acknowledged or the accumulated errors exceed RetryMax, a constant private to the Xmodm Unit. A timeout status is returned if errors were exceeded, a transmission complete status if EOT was properly acknowledged.

l. **Send_CAN**

(1) Type: Procedure

(2) Purpose: Used to inform the other side of the communications link that the Xmodem operation is to be aborted.

(3) Description of Parameters:

Input: None.

Output: Two CAN characters are sent out the communications port.

(4) Subroutines Called:

Xmodm.WriteAux

(5) Process Description

Two CAN characters are sent out the communications port.

m. **Update_Status**

(1) Type: Procedure

(2) Purpose: To display or refresh the current status of the calling program in a monitor window.

(3) Description of Parameters:

Input: Typically this procedure writes current information on the status of a data transfer, the number of bytes and blocks sent or received, and the count of the number of errors accumulated on the transaction in a formatted display.

Output: A window display of the current status.

(4) Subroutines Called:

Wndow.Get_Window

CRT.GoToXY

(5) Process Description

This process is used several places in this unit, and operates identically in each. The process depends on the caller to open a properly sized window and to set a variable called Status_ID to allow the status window to be accessed via Get_Window. Once reopened, the procedure writes the current status information using variables local to the caller. The procedure then resets the working window to that of the caller's Monitor_ID.

n. **Xmodem_Xfer**

(1) Type: Function

(2) Purpose: Perform an Xmodem file transfer.

(3) Description of Parameters:

Input: Send, TRUE to send a file, FALSE to receive; and Blocksize, the size of the data buffer to use.

Output: A status code indicating success or what problem was encountered.

(4) Subroutines Called:

Update.Status (local to this function)
CRT.ClrScr
CRT.Delay
CRT.GoToXY
CRT.KeyPressed
CRT.ReadKey
DataCom.RS_Eight_Bits
General.Beep
System.BlockWrite
System.Assign
System.Reset
System.Rewrite
Xmodm.Sync_Send
Xmodm.Get_Buffer
Xmodm.Send_Record
Xmodm.Sync_Receive
Xmodm.WriteAux
Wndow.Close_Window
Wndow.Open_Window

(5) Process Description

The public variable Monitor_Transfers is checked to see if a monitor window is to be opened to display the characters transfered. If TRUE, the window and a monitor file are opened. The status window is then opened and unchanging field names written. RS_Eight_Bits is called to insure the communications port passes eight bit data, regardless of its settings. After initializing the variables used to report status, the function branches depending on whether a file is to be sent or received. If Send is TRUE, Sync_Send is called to detect sync characters from the receiver. If Sync_Send times out, the transfer is aborted and the timeout is reported to the caller. IF sync is detected, file buffers are obtained from Get_Buffer and sent via Send_Record until EOF is detected or too many errors are encountered. If successful, EOT is sent to the receiver to signal the end of transmission. The KeyPressed function is monitored at several points, and will cause an immediate abort with status returned to the caller. If Send is FALSE, Sync_Receive is called to send sync characters. If a timeout is not encountered, Receive_Record is called repeatedly to obtain received buffers and monitor status. The transfer terminates on receipt of EOT (competion), too many errors detected or a keypress indication, with appropriate status returned to the caller. Update_Status is called several times throughout each branch to indicate progress or report errors. The transfer file is then closed, as are the monitor and status windows. RS_Initialize is called to reset the communications port to its previous word length.

o. Command_Xfer

(1) Type: Function

(2) Purpose: Transfer a single command packet.

(3) Description of Parameters:

Input: Send, TRUE to send a packet, FALSE to receive a packet; Buf, the data buffer send or received; Blocksize, the size of the data buffer.

Output: A status code indicating success or what problem was encountered.

(4) Subroutines Called:

Update.Status (local to this function)
CRT.ClrScr
CRT.Delay
CRT.GoToXY
CRT.KeyPressed
CRT.ReadKey
General.Beep
Xmodm.Sync_Send
Xmodm.Get_Buffer
Xmodm.Send_Record
Xmodm.Sync_Receive
Xmodm.WriteAux
Wndow.Close_Window
Wndow.Open_Window

(5) Process Description

This function operates similarly to Xmodem_Xfer, except that a single Xmodem packet is transferred. The public variable Monitor_Transfers is checked to see if a monitor window is to be opened to display the characters transfered. If TRUE, the window and the monitor file are opened. The status window is then opened and unchanging field names written. RS_Eight_Bits is called to insure the communications port passes eight bit data, regardless of its settings. After initializing the variables used to report status, the function branches depending on whether a file is to be sent or received. If Send is TRUE, Sync_Send is called to detect sync characters from the receiver. If Sync_Send times out, the transfer is aborted and the timeout is reported to the caller. IF sync is detected buf is sent via Send_Record. If successful, EOT is sent to the receiver to signal the end of transmission. The KeyPressed function is monitored at several points, and will cause an immediate abort with status returned to the caller. If Send is FALSE, Sync_Receive is called to send sync characters. If a timeout is not encountered, Receive_Record is called to obtain received buffer and monitor status. The transfer terminates on receipt of EOT (competion), too many errors detected or a keypress indication, with appropriate status returned to the caller. Update_Status is called several times throughout each branch to indicate progress or report errors. The monitor file is then closed, as are the monitor and status windows. RS_Initialize is called to reset the communications port to its previous word length.

p. Transfer_File

(1) Type: Procedure

(2) Purpose: To obtain the name of the file to be transferred from the local operator.

(3) Description of Parameters:

Input: Send, TRUE if a file send is desired, FALSE to receive a file.

Output: Monitor display.

(4) Subroutines Called:

Wndow.Open_Window
Wndow.Close_Window
Support.NoFile
System.Assign
System.Length
System.Reset
System.Rewrite
System.Upcase

(5) Process Description

Transfer_File first opens a window to ask the operator what filename is to be transferred. The transfer is aborted and NoFile is called if the file is not found or cannot be opened. Depending on Send, the file is opened for reading or writing and then Xmodem_Xfer is called to accomplish the transfer.

q. Respond_by_File

(1) Type: Procedure

(2) Purpose: To allow the remote Slave to send the results of a program or other message contained in a file to the Master.

(3) Description of Parameters:

Input: Response, the file to be sent.

Output: None from this procedure.

(4) Subroutines Called:

Wndow.Open_Window
Wndow.Close_Window
System.Assign
System.Length
System.Reset
System.Rewrite
System.Upcase

(5) Process Description

Transfer_File first opens a window to ask the operator what filename is to be transferred. The transfer is aborted if the file is not found or cannot be opened. Depending on Send, the file is opened for reading or writing and then Xmodem_Xfer is called to accomplish the transfer.

r. Get_Response

(1) Type: Function

(2) Purpose: To allow the Master to receive file responses from a program completed by the Slave.

(3) Description of Parameters:

Input: BlockSize, the size of the Xmodem buffers.

Output: Status code of the call.

(4) Subroutines Called:
CRT.KeyPressed
CRT.ReadKey
DataCom.RS_Eight_Bits
DataCom.RS_Restore
Xmodm.Sync_Receive
Xmodm.Receive_Record
Xmodm.WriteAux
System.Assign
System.Close
System.Rewrite
Wndow.TextColor
Wndow.TextBackGround

(5) Process Description

For this function, the monitor window is set to the current window, and the monitor file is directed to NUL, the bit bucket. This satisfies ReadAux and WriteAux so that the display will operate properly without creating an unnecessary file. RS_Eight_Bits is called to insure the communications port passes eight bit data, regardless of its settings. After initializing the variables used to report status, Sync_Receive is called to send sync characters. If a timeout is not encountered, Receive_Record is called to obtain received buffer and monitor status. The transfer terminates on receipt of EOT (competion), too many errors detected or a keypress indication, with appropriate status returned to the caller. Update_Status is called several times throughout each branch to indicate progress or report errors. RS_Initialize is called to reset the communications port to its previous word length, and the dummy monitor file is closed.

s. **Xmodm Unit Initialization**

(1) Type: Unit Initialization Procedure

(2) Purpose: To initialize the unit on loading.

(3) Description of Parameters:
Input: Suppress_EOT, Monitor_Transfers.
Output: Suppress_EOT, Monitor_Transfers.

(4) Subroutines Called: None.

(5) Process Description

Suppress_EOT and Monitor_Transfers are set to their default values.

{

APPENDIX P

SOURCE LISTING FOR UNIT DATACOM

```
}
(*************************************************************************)
(****                          DATACOM.PAS                            ****)
(****  This is the unit that accomplishes all interface to the  ****)
(****  communications ports for character, string and buffer    ****)
(****  transfer.  It also initializes the communications ports  ****)
(****  and provides interrupt interrupt service routines for    ****)
(****  character receive.                                       ****)
(****                                                           ****)
(****  References:                                              ****)
(****    Interface:   Edwards, C.G., Advanced Techniques in     ****)
(****                 Turbo Pascal, pp. 220 - 238, Sybex,       ****)
(****                 Inc., 1987.                               ****)
(****                                                           ****)
(****    Multiple                                               ****)
(****    Ports:       Kimura, N., <abcscnuk@csuna.uucp>,        ****)
(****                 info-pascal-@vim.brl.mil message,         ****)
(****                 Subject:  Re: TP4.0 Aux Problem,          ****)
(****                 Message-ID:  <1376@csuna.uucp>,           ****)
(****                 17 Nov 88 10:20:54 GMT.                   ****)
(****                                                           ****)
(****    Low Level                                              ****)
(****    Procedures:  de Boer, R., <reino@euraiv1.uucp>,        ****)
(****                 info-pascal-@vim.brl.mil message,         ****)
(****                 Subject:  Serial Unit in TP4,             ****)
(****                 Message-ID:  <797@euraiv1.uucp>,          ****)
(****                 15 Nov 88 14:17:15 GMT.                   ****)
(****                                                           ****)
(****   UART/PIC                                                ****)
(****   Declarations: Greenberg, R.M., "TSRCOMM, a Replacement  ****)
(****                 for Interrupt 14", source listing,        ****)
(****                 Ross M. Greenberg, 1987.                  ****)
(****                                                           ****)
(****  Developed by Nelson Ard.                                 ****)
(****                                                           ****)
(****  Last modification Sep 89.                                ****)
(*************************************************************************)

(* Modification history

   8 Sep 89 - added RS_Eight_Bits to change the port data work
              width to eight bits for Xmodem protocol operation.

*)
```

```
UNIT DATACOM;

INTERFACE

USES General, CRT, Dos;

CONST
      COM1 = 1;
      COM2 = 2;
      COM3 = 3; {not implemented, but MS-DOS knows about them}
      COM4 = 4; {not implemented, but MS-DOS knows about them}

(************************  Start Edwards Excerpt  *********************)
TYPE
     RS_Baud = (B110,B150,B300,B600,B1200,B2400,B4800,B9600,B19200,
                B38400 );
     RS_Parity = (None,Odd,Nevermind,Even);

     RS_Config = Record
                   Stop,
                   Length : byte;
                   Alias  : string[10];
                   Speed  : RS_Baud;
                   Parity : RS_Parity;
                   IRQNo  : byte;
                   Installed : boolean;
                 end; { RECORD }

     PortRange = COM1..COM2;

VAR
    Current_Com : Byte; {public, specifies current port for
                         command or file transfer}
    ComPort : ARRAY [ PortRange ] OF RS_Config;


Procedure RS_Break;
{ This procedure instructs the currently selected data communications
  port to send a break signal}

Function RS232_Avail:Boolean;
{This function returns TRUE if there are characters to be read from
 the RS232 port. It is analogous to the Turbo function KEYPRESSED for
 the keyboard.
}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
Function RS232_In:Char;
{The AUX device is set to point to this function for input. It returns
the next character received from the RS232 port.
}

Procedure RS232_Out ( Param : Char );

{ Sends the character to the RS232 port. }

Procedure RS_Initialize(Com:Byte;Speed:RS_Baud;Parity:RS_Parity;
                        Stop,Length:Byte);

{ Initialize communications port.  Vector the appropriate interrupt to
  point to our interrupt service routine.  Initialize hardware
  handshaking lines.  Store current settings in a data structure for
  restoration.

 Input: COM    - The RS232 port to be handled
        Speed  - The baud rate of the line
        P      - The parity of the line
        Stop   - The number of stop bits
        Length - The number of data bits
}

Procedure RS_Cleanup;

{This procedure should be called on exit to disable interrupts on the
 RS232 port and reset everything to its default state.
}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  End Edwards Excerpt  **********************)

Procedure PurgeLine;
{ This function clears the receive buffer and UART receive buffer for
  the currently selected port}

Function Connected : boolean;
{ Returns TRUE if the Data Set Ready line is true, signalling hardware
  handshaking }

Function RS232_peek : Char;

{ Added to allow nondestructive read of the currently selected port
  input buffer for xmodm.Sync_receive}
```

```
Procedure Send_String ( S : String );

{ Send a string out the currently selected RS232 port }

Procedure RS_Restore ( COM : byte );

{ Reinitialize the COM Port }

Procedure RS_Eight_Bits;

{ Adjust the comport for eight bits regardless of current setting }
```

```
IMPLEMENTATION

(************************ Start Greenberg extract **********************)
CONST
  { UART declarations }
  { Interrupt Enable Register }
  { Or one or more of these bits to enable the respective interrupts }

  IER_RDA  = $01;  { Receive Data Available Int Bit      ---- ---1 }
  IER_THRE = $02;  { Transmitter Hold Register Empty Bit ---- --1- }
  IER_RLS  = $04;  { Receive Line Status Int Bit         ---- -1-- }
  IER_MS   = $08;  { Modem Status Int Bit                ---- 1--- }

  { Interrupt Identification Register }
  { Check the lower four bits to see what interrupt called }

  IIR_RLS  = $05;  { Receiver Line Status Interrupt      ---- -101 }
  IIR_RDA  = $04;  { Receive Data Available              ---- -100 }
  IIR_THRE = $02;  { Transmitter Hold Register is Empty  ---- -010 }
  IIR_PEND = $01;  { zero if * any * interrupt pending   ---- -001 }
  IIR_MS   = $00;  { Modem Status interrupt              ---- -000 ]

  { Line Control Register }
  { Or one or more of these bits to select comm port parameters }

  LCR_CHR5     = $00;  { Five bit character              ---- --00 }
  LCR_CHR6     = $01;  { Six bit character               ---- --01 }
  LCR_CHR7     = $02;  { Seven bit character             ---- --10 }
  LCR_CHR8     = $03;  { Eight bit character             ---- --11 }
  LCR_STOP1    = $00;  { One stop bit                    ---- -0-- }
  LCR_STOP2    = $04;  { Two stop bits                   ---- -1-- }
  LCR_NOPARITY = $00;  { No parity                       ---- 0--- }
  LCR_PARITYEN = $08;  { Enable parity (see SPARITY and
                                          EPARITY        ---- 1--- }
  LCR_EPARITY  = $10;  { Even parity bit                 ---1 ---- }
  LCR_SPARITY  = $20;  { Stick parity                    --1- ---- }
  LCR_BREAK    = $40;  { Transmits a BREAK (space)       -1-- ---- }
  LCR_DLAB     = $80;  { Divisor Latch Access bit        1--- ---- }
```



```
********************** Continue Greenberg Excerpt  *********************)
```

```
(********************** Continue Greenberg extract **********************)

  { Modem Control Register }
  { Or one or more of these bits to signal the modem }

  MCR_DTR      = $01;  { set Data Terminal Ready          ---- ---1 }
  MCR_RTS      = $02;  { set Request To Sent              ---- --1- }
  MCR_OUT1     = $04;  { Output 1 (resets Hayes modem)    ---- -1-- }
  MCR_OUT2     = $08;  { Output 2 (allows comm
                                  port interrupts)       ---- 1--- }
  MCR_LOOP     = $10;  { Loopback test                    ---1 ---- }

  { Line Status Register }
  { Test one or more of these bits to determine comm port status }

  LSR_DATA     = $01;  { data is available                ---- ---1 }
  LSR_OVERRUN  = $02;  { overrun error bit                ---- --1- }
  LSR_PARITY   = $04;  { parity error bit                 ---- -1-- }
  LSR_FRAMING  = $08;  { framing error bit                ---- 1--- }
  LSR_BREAK    = $10;  { BREAK detected bit               ---1 ---- }
  LSR_THRE     = $20;  { Transmit Holding Register Empty  --1- ---- }
  LSR_TSRETY   = $40;  { Transmit Shift Register Empty    -1-- ---- }

  { Modem Status Register }
  { Test one or more of these bits to determine modem actions }

  MSR_DEL_CTS  = $01;  { delta Clear To Send              ---- ---1 }
  MSR_DEL_DSR  = $02;  { delta Data Set Ready             ---- --1- }
  MSR_EDGE_RI  = $04;  { Trailing Edge of Ring Indicator  ---- -1-- }
  MSR_DEL_SIGD = $08;  { delta Receive Line Signal Det    ---- 1--- }
  MSR_CTS      = $10;  { Clear To Send                    ---1 ---- }
  MSR_DSR      = $20;  { Data Set Ready                   --1- ---- }
  MSR_RI       = $40;  { Ring Indicator - entire ring     -1-- ---- }
  MSR_DCD      = $80;  { Data Carrier Detect - on line    1--- ---- }
```



```
************************  End Greenberg Excerpt  **********************)
```

```
(******************** Start Edwards Excerpt ***************************)

  { IRQ Lines }
  IRQline      : ARRAY [ PortRange ] OF byte = ( 4, 3 );

TYPE

  INS8250 = record
            THR : word;  { Transmit Holding Register }
            RBR : word;  { Receive Holding Register  }
            IER : word;  { Interrupt Enable Register }
            IIR : word;  { Interrupt Ident  Register }
            LCR : word;  { Line Control Register     }
            MCR : word;  { Modem Contro Register     }
            LSR : word;  { Line Status Register      }
            MSR : word;  { Modem Status Register     }
            DLL : word;  { Divisor Latch LSB         }
            DLM : word;  { Divisor Latch MSB         }

            END;

CONST
      RS_Buffer_Size = 4095; {Size of Buffer - 1...Change this if you
                              want a different buffer size}

      { 8259 PIC declarations }
      ISR       = $20; { Interrupt Service Register }
      IMR       = $21; { Interrupt Mask Register }
      IRQ4_Mask = $EF; { Enable for COM1 }
      IRQ3_Mask = $F7; { Enable for COM2 }

      { IBM PC comm port interrupt vectors }
      COM1_INTR = $0C;
      COM2_INTR = $0B;

      RS_Error : byte = 0;
      Chk_DSR  : boolean = FALSE;
      Chk_CTS  : boolean = FALSE;

    Regs  : Array [1..2] of INS8250 =
          ((THR:$3F8; RBR:$3F8; IER:$3F9; IIR:$3F9; LCR:$3FB;
            MCR:$3FC; LSR:$3FD; MSR:$3FE; DLL:$3F8; DLM:$3F9),
           (THR:$2F8; RBR:$2F8; IER:$2F9; IIR:$2F9; LCR:$2FB;
            MCR:$2FC; LSR:$2FD; MSR:$2FE; DLL:$2F8; DLM:$2F9));

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
Var RS_Buffer    : Array [1..2,0..RS_Buffer_Size] of Byte;
    RS_Buf_Head,
    RS_Buf_Tail  : Array [1..2] OF word;
    index : byte;
    Line_settings : byte;

Procedure DisableInterrupts;

{ Insert assembly code to disable computer interrupts }

INLINE ( $FA );

Procedure EnableInterrupts;

{ Insert assembly code to enable computer interrupts }

INLINE ( $FB );

Function RS232_Avail:Boolean;
{This function returns TRUE if there are characters to be read from
 the RS232 port. It is analogous to the Turbo function KEYPRESSED for
 the keyboard.
}
     Begin
       Rs232_Avail :=
         RS_Buf_Head  [ Current_COM ] <> RS_Buf_Tail  [ Current_COM ];
     End;  {or RS232_Avail}

Procedure RS232_ISR1
  (Flags, CS, IP, AX, BX, CX, DX, SI, DI, DS, ES, BP : word);
  INTERRUPT;

{ This procedure handles interrupts from RS232 port one
  THIS PROCEDURE MUST NOT BE CALLED BY ANY OTHER PROCESS }

Begin
  DisableInterrupts;
  RS_Error:=Port[ Regs[ COM1 ].LSR ] and $1E;
  RS_Buffer[ COM1, RS_Buf_Tail [ COM1 ]] := Port[Regs[ COM1 ].RBR];
  RS_Buf_Tail[ COM1 ] := ( RS_Buf_Tail[ COM1 ]+1)
    mod (RS_Buffer_Size+1);
  EnableInterrupts;
  Port[ $20 ] := $20;   {Report end of service to PIC}
End;  {of RS232_ISR1 }

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Procedure RS232_ISR2
  (Flags, CS, IP, AX, BX, CX, DX, SI, DI, DS, ES, BP : word);
  INTERRUPT;

{This procedure handles interrupts from RS232 port two
  THIS PROCEDURE MUST NOT BE CALLED BY ANY OTHER PROCESS}

Begin
  DisableInterrupts;
  RS_Error:=Port[ Regs[ COM2 ].LSR ] and $1E;
  RS_Buffer[ COM2, RS_Buf_Tail [ COM2 ]] := Port[Regs[ COM2 ].RBR];
  RS_Buf_Tail[ COM2 ] := ( RS_Buf_Tail[ COM2 ]+1)
    mod (RS_Buffer_Size+1);
  EnableInterrupts;
  Port[ $20 ] := $20;   {Report end of service to PIC}
End;  {of RS232_ISR2 }

Procedure RS_Break;
{ This procedure instructs the currently selected data communications
  port to send a break signal}

Begin
   Port[Regs[Current_Com].LCR] :=
     Port[Regs[Current_Com].LCR] or LCR_BREAK;
   Delay(200);   {1/5 second}
   Port[Regs[Current_Com].LCR] :=
     Port[Regs[Current_Com].LCR] xor LCR_BREAK;
End;  {of RS_Break}

Function RS232_In:Char;
{The AUX device is set to point to this function for input. It returns
the next character received from the RS232 port.
}

Begin
  While RS_Buf_Head [ Current_COM ] = RS_Buf_Tail [ Current_COM ] Do
    Delay(10);
  RS232_In :=
    Char ( RS_Buffer [ Current_COM, RS_Buf_Head [ Current_COM ]]);
  RS_Buf_Head [ Current_COM ] :=
    (RS_Buf_Head [ Current_COM ]+1) mod (RS_Buffer_Size+1);
End;  {of RS232_In}


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Procedure RS232_Out ( Param : Char );

{ Sends the character to the RS232 port. }

Begin
     While (( Port [Regs [ Current_Com ].LSR] and $20) <> $20)
     {Transmit Reg empty}
        do Delay(1);
     (* Request to send *)
     Port [ Regs [ Current_COM ].MCR ] := MCR_RTS OR MCR_OUT2;
     IF Chk_DSR THEN
     While ((Port[ Regs [ Current_COM ].MSR] and MSR_DSR) <> MSR_DSR)
        do Delay(1);   {Wait a while}
     IF Chk_CTS THEN
     While ((Port[ Regs [ Current_COM ].MSR] and MSR_CTS) <> MSR_CTS)
        do Delay(1);   {Wait a while}
     Port[Regs[ Current_COM ].THR] := Byte ( Param );
     RS_Error:=0;
End;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
************************  End Edwards Excerpt  ***********************)

(************************** Start de Boer extract *********************)
PROCEDURE Enable ( IRQ : byte );

{ Set the Interrupt Mask Register on the Programmable Interrupt
  Controller to recognize interrupts from this port }

BEGIN
  Port [IMR] := Port [IMR] AND NOT ( 1 SHL IRQ );
END; { Enable }

PROCEDURE Disable ( IRQ : byte );

{ Reset the Interrupt Mask Register on the Programmable Interrupt
  Controller to ignore interrupts from this port }

BEGIN
  Port [IMR] := Port [IMR] OR ( 1 SHL IRQ );
END; { Disable }


(*  Reprinted from "Serial Unit in TP4" by Reino de Boer, by permission
    of the author.  Copyright 1987 Reino de Boer.  All rights reserved.
************************ Continue Boer Excerpt  **********************)
```

```
(************************ Continue de Boer extract ********************)

PROCEDURE Establish ( COM : byte );

{ Raise all hardware handshaking lines to prepare for
  communications }

BEGIN
  WITH Regs [ COM ] DO
    Port [ MCR ] := MCR_DTR OR MCR_RTS OR MCR_OUT2;
END;

PROCEDURE SendEOI ( IRQ : Byte );

{ Send an End Of Interrupt command to the Programmable Interrupt
  Controller to let it know we are done servicing this interrupt }

BEGIN
  Port [ ISR ] := $60 OR IRQ;
END;

Procedure ResetChip ( Com : Byte );

{ Disable UART generated interrupts, drop the hardware handshaking
  lines.  Shut down the currently selected communications port }

Var Dummy : byte;

Begin
  WITH Regs [ Com ], Comport [ Com ] DO BEGIN
    WHILE ( ( Port [ LSR ] AND LSR_DATA ) <> 0 ) DO
      Dummy := Port [ RBR ];
    DisableInterrupts;
    { Allow none of the interrupt types }
    Port [ IER ] := 0;
    { Tell modem we're not ready }
    Port [ MCR ] := Port [ MCR ] AND
      NOT ( MCR_OUT2 OR MCR_DTR OR MCR_RTS );
    { Disable all interrupts for this port }
    Disable ( IRQNo );
    EnableInterrupts;
  END;
END;

CONST { Bit rate divisor table }
      Divisor : ARRAY [RS_Baud] OF word =
                ( 1047, 768, 384, 192, 96, 48, 24, 12, 6, 3 );

(*  Reprinted from "Serial Unit in TP4" by Reino de Boer, by permission
    of the author.  Copyright 1987 Reino de Boer.  All rights reserved.
************************ Continue Boer Excerpt  ********************)
```

```
(************************ Continue de Boer extract ********************)

{ Select bit rate by programming the PBRG }
PROCEDURE SelectBitRate( COM : byte; Speed : RS_Baud );

CONST PBRG_Settle : word = 250;

VAR BaudDiv : word;

BEGIN
  { Update port data }
  ComPort [ Com ].Speed := Speed;
  BaudDiv := Divisor [ Speed ];
  { Set Divisor Latch Access Bit }
  port[ Regs [ Com ].LCR ] :=
    port[ Regs [  COM ].LCR ] OR LCR_DLAB;
  { Bit rate divisor to PBRG }
  portw[ Regs [  COM ].RBR ] := BaudDiv;
  { Give port some time to settle }
  delay( PBRG_Settle );
  { Reset function of RBR }
  port[ Regs [  COM ].LCR ] :=
    port[ Regs [  COM ].LCR ] XOR LCR_DLAB;
END; { SelectBitRate }

{ Set word length in Line Control Register }
PROCEDURE SelectWordLength( COM : Byte; Length : byte );

VAR LineControl : byte;

BEGIN
  { Update port data }
  ComPort [ Com ].Length := Length;
  LineControl := port[ Regs [ Com ].LCR ];
  LineControl := ( LineControl AND ( NOT LCR_CHR8 ) )
                 OR (Length - 5);
  { Set relevant bits }
  port[ Regs [  COM ].LCR] := LineControl;
END; { SelectWordLength }
```


```
************************ Continue Boer Excerpt  ********************)
```

```
(************************ Continue de Boer extract ********************)

{ Set stopbits in Line Control Register }
PROCEDURE SelectFraming( COM : Byte; Stop : byte );

VAR LineControl : byte;

BEGIN
  { Update port data }
  ComPort [ Com ].Stop := Stop;
  LineControl := port[ Regs [ Com ].LCR ];
  LineControl := ( LineControl AND ( NOT LCR_Stop2 ) )
    OR ((Stop - 1)*4);
  { Set relevant bits }
  port[ Regs [ COM ].LCR ] := LineControl;
END; { SelectFraming }


{ Set parity in Line Control Register }
PROCEDURE SelectParity( COM : byte; Parity : RS_Parity);

VAR LineControl : byte;

BEGIN
  ComPort[ Com ].Parity := Parity;
  { Update port data }
  LineControl := port[ Regs [ Com ].LCR ];
  LineControl := ( LineControl AND ( NOT $40 ) )
                  OR ORD(Parity)*8;
  { Set relevant bits }
  port[ Regs [ COM ].LCR] := LineControl
END; { SelectParity }

CONST RTS_Settle   : byte = 2;
      DTR_Settle   : byte = 2;
      PBRG_Settle  : word = 250;

(*  Reprinted from "Serial Unit in TP4" by Reino de Boer, by permission
    of the author.  Copyright 1987 Reino de Boer.  All rights reserved.
*************************  End de Boer Excerpt  ********************)
```

```
Procedure PurgeLine;
{ This function clears the receive buffer and UART receive buffer for
  the currently selected port}

VAR
  Dummy : Byte;

BEGIN
  RS_Buf_Head [ Current_COM ] := 0;
  RS_Buf_Tail [ Current_COM ] := 0;
  Dummy := Port[Regs[Current_COM].RBR];
End;  {of PurgeLine}

FUNCTION Connected : boolean;
{ Returns TRUE if the Data Set Ready line is true, signalling hardware
  handshaking }

BEGIN
  Connected := Port[Regs[Current_Com].MSR] and $80 = $80;
END;

Function RS232_peek : Char;

{ Added to allow nondestructive read of the currently selected port
  input buffer for xmodm.Sync_receive}

Begin
     While RS_Buf_Head [ Current_COM ] =
       RS_Buf_Tail [ Current_COM ] do Delay(10);
     RS232_peek := Char( RS_Buffer[ Current_COM,
                         RS_Buf_Head [Current_COM] ]);
End;  {of RS232_Peek}

Procedure RS_Eight_Bits;

{ Adjust the comport for eight bits regardless of current setting }

BEGIN
  Port [ Regs [ Current_Com ].LCR ] := LCR_NOPARITY OR LCR_STOP1
                                       OR LCR_CHR8;
END;

Procedure RS_Restore ( COM : byte );

{ Reinitialize the COM Port }

BEGIN
  WITH Comport [ COM ] DO
    RS_Initialize( Com, Speed, Parity, Stop, Length );
END;
```

```
Procedure Send_String ( S : String );

{ Send a string out the currently selected RS232 port }

BEGIN
  IF Length (S) > 0 THEN
    FOR index := 1 to Length (S) DO
      RS232_Out ( S [ index ] );
END;

(*********************  Start Edwards Excerpt ************************)
Procedure RS232_Init ( COM, Params : word );

{ Call BIOS interrupt $14 with a formatted word to initialize the
  currently selected communications port }

VAR Regs : DOS.Registers;

BEGIN
  Regs.DX := Com-1;
  Regs.AX := Params;
  Intr($14,Regs);
END;

Procedure RS_Initialize(Com:Byte;Speed:RS_Baud;Parity:RS_Parity;
                        Stop,Length:Byte);

{ Initialize communications port.  Vector the appropriate interrupt to
  point to our interrupt service routine.  Initialize hardware
  handshaking lines.  Store current settings in a data structure for
  restoration.

 Input: COM    - The RS232 port to be handled
        Speed  - The baud rate of the line
        P      - The parity of the line
        Stop   - The number of stop bits
        Length - The number of data bits
}

Var Params : word;


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
**********************  Continue Edwards Excerpt  **********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
Begin
  WITH Regs [ COM ] DO BEGIN
     Current_Com:=Com; {save comm port in local variable}
     Params := Ord(Speed)*32 + Ord(Parity)*8 + (Stop-1)*4 + Length-5;
     { Calling the BIOS service to initialize the port
       * clears * all UART interrupts }
     RS232_Init ( COM, Params );
     Delay ( PBRG_Settle ); { delay to allow UART to settle }
     Port [ LCR ] :=
       Port [ LCR ] AND ( NOT LCR_DLAB );
     { Set our interrupt handler }

     CASE Com OF

       1 : SetIntVec ( COM1_INTR, Addr( RS232_ISR1 ));
       2 : SetIntVec ( COM2_INTR, Addr( RS232_ISR2 ));

     END;
     ResetChip ( Com );
     DisableInterrupts;
     Establish ( COM );
     Enable ( Comport [ Current_Com ].IRQNo );
     { Interrupt on receive only }
     Port [ Regs [ COM ].IER ] := IER_RDA;
     { Clear the port buffer }
     RS_Buf_Head [ Com ] :=0;
     RS_Buf_Tail [ Com ] :=0;
     { Reset any stray interrupts in the PIC }
     SendEOI ( Comport [ Current_Com ].IRQNo );
     EnableInterrupts;

       Comport [ Current_Com ].Speed  := Speed;
       Comport [ Current_Com ].Parity := Parity;
       Comport [ Current_Com ].Stop   := Stop;
       Comport [ Current_Com ].Length := Length;
       Comport [ Current_Com ].Installed := TRUE;
  END;
End;  {of Initialize}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
    Modified after "Serial Unit in TP4" by Reino de Boer, reprinted by
    permission of the author.  Copyright 1987 Reino de Boer.  All
    rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
CONST
  ExitPtr : pointer = NIL;
  OldIntVec1 : pointer = NIL;
  OldIntVec2 : pointer = NIL;
  Old_IMR    : byte = 0;
  Old_IER1   : byte = 0;
  Old_IER2   : byte = 0;

Procedure RS_Cleanup;

{This procedure should be called on exit to disable interrupts on the
 RS232 port and reset everything to its default state.
}

Begin
  Comport [ Current_Com ].Installed := FALSE;
  ResetChip ( Current_Com );
End;  {of Cleanup}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
    Modified after "Serial Unit in TP4" by Reino de Boer, reprinted by
    permission of the author.  Copyright 1987 Reino de Boer.  All
    rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
(* This is the error handler for Datacomm *)

(*****************************  Start Swan Excerpt  ******************)
CONST
  HexDigit : ARRAY [0..15] OF Char = '0123456789ABCDEF';

TYPE
  string2 = string[2];
  string4 = string[4];

  PtrRec = RECORD
             Ofs, Seg : word;
           END;

  FUNCTION HexByte (B : Byte ) : string2;
  BEGIN
    HexByte := HexDigit [B SHR 4] + HexDigit[B AND $F];
  END;

  FUNCTION Hex (I : Word) : string4;

  BEGIN
    Hex := HexByte(Hi(I)) + HexByte(Lo(I));
  END;
(*  Reprinted from Mastering Turbo Pascal Files By Tom Swan, by
    permission of Howard W. Sams and Company.  Copyright 1987 Howard W.
    Sams and Company.  All rights reserved.
*****************************  End Swan Excerpt  **********************)

{$F+}  PROCEDURE Datacomm_Error; {$F-}

{ This is the Exit Procedure for * this * unit }

VAR index : byte;

  BEGIN
    IF (ExitCode <> 0) OR (ErrorAddr <> NIL) THEN
      BEGIN
        Assign (Output, '');
        Rewrite(OutPut);
        (*Writeln(#7);*)
        IF ExitCode = $FF THEN
          Writeln('USER BREAK')
        ELSE

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
          BEGIN
            Writeln ('Critical Error # ', HEX(ExitCode));
            Write('AT PROGRAM LOCATION');
            Writeln(HEX(seg(ErrorAddr^)),':', Hex(ofs(ErrorAddr^)));
          END;
      END;
      DisableInterrupts;
      { Restore the previous interrupt vectors }
      SetIntVec ( COM1_INTR, OldIntVec1 );
      SetIntVec ( COM2_INTR, OldIntVec2 );
      EnableInterrupts;
      { Shut down the ports }
      FOR index := COM1 TO COM2 DO BEGIN
        Port[ Regs [ index ].LCR]:=Port[ Regs [ index ].LCR] and $7F;
        Port[ Regs [ index ].IER]:=0;
        Port[ Regs [ index ].MCR]:=0;
      END;
      { Restore the PIC interrupt mask }
      Port [ IMR ] := Old_IMR;
      ExitProc := ExitPtr;
  END; { Datacomm_Error }

BEGIN { Unit Initialization }
  CheckBreak := TRUE;
  { Save the existing exit procedure for this unit }
  ExitPtr := ExitProc;
  { Save the existing interrupt mask for the PIC }
  Old_IMR := Port [ IMR ];
  { Save the current serial port interrupt vectors }
  GetIntVec ( COM1_INTR, OldIntVec1);
  GetIntVec ( COM2_INTR, OldIntVec2);
  { Clear the receive buffers }
  RS_Buf_Head [ COM1 ] := 0;
  RS_Buf_Head [ COM1 ] := 0;
  RS_Buf_Tail [ COM2 ] := 0;
  RS_Buf_Tail [ COM2 ] := 0;
  { link in our unit exit procedure to undo all of the above on
    program termination }
  ExitProc := Addr(Datacomm_error);
  { Set up both ports to initial values }



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
    Modified after "Re:: TP4.0 Aux Problem" by Naoto Kimura, reprinted
    by permission of the author.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
  FOR index := COM1 TO COM2 DO
    WITH Comport[index] DO BEGIN
      Stop   := 1;
      Length := 8;
      Alias  := '';
      Speed  := B4800;
      Parity := None;
      IRQNo  := IRQLine [ index ];
      Installed := FALSE;
    end; { COMPORT initializaton }
END.
```

```
(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.

    Modifications reprinted from "Serial Unit in TP4" by Reino de Boer,
    by permission of the author.  Copyright 1987 Reino de Boer.  All
      rights reserved.
(************************  End Edwards Excerpt  ************************)
```

{

APPENDIX Q

SOURCE LISTING FOR UNIT DIRECTOR

```
}
(*************************************************************************)
(****                          DIRECTOR.PAS                           ****)
(*************** Turbo Pascal 4.0 unit to read a directory *********)
(*************** and display it                            *********)
(*************** Date: 28 July 1989, 10:06:53 MEZ          *********)
(*************** From: Christian Boettger                  *********)
(*************** +49 (0)531 3915113 / I2010506 at DBSTU1   *********)
(*                                                                   *)
(*               Modified slightly to change presentation            *)
(*               for the window manager                              *)
(*               and to use the error reporting capability           *)
(*               of UNIT ErrorCod                                    *)
(*               by Nelson Ard                                       *)
(*               Last modification Sep 89                            *)
(*************************************************************************)

unit director;

interface

uses dos,crt, ErrorCod; {ErrorCod added}

procedure ViewDir(MatchPtrn : string; FromLine, ToLine : integer);
procedure ShowDir(MatchPtrn : string; FromLine, ToLine : integer;
                  var error : integer);

(*  Reprinted from "Turbo Pascal 4.0 unit to read a directory and
    display it" by Dipl. Phys. Christian Boettger, by permission of the
    author.
********************** Continue Boettger Excerpt  *********************)
```

```
(********************* Continue Boettger Excerpt  *********************)
implementation

procedure StandBy;
   var x,y   : integer;
       muell : char;
   begin
     x:=whereX; y:= WhereY;
     HighVideo;
     write('Hit any key to continue ');
     NormVideo;
     repeat until keypressed;
     muell := ReadKey;
     write('                        ');
     GotoXY(x,y);
   end;

(*  Reprinted from "Turbo Pascal 4.0 unit to read a directory and
    display it" by Dipl. Phys. Christian Boettger, by permission of the
    author.
*************************  End Boettger Excerpt  *********************)
```

```
(********************** Start Verbraeck Excerpt  ********************)

procedure ViewDir(MatchPtrn : string; FromLine, ToLine : integer);

(*************************************************************************
-------------------------------------------------------------------------
Ir. Alexander Verbraeck                          e-mail:
Delft University of Technology                   winfave@hdetud1.bitnet
Department of Information Systems                winfave@dutrun.uucp
PO Box 356, 2600 AJ  The Netherlands
-------------------------------------------------------------------------
*************************************************************************)

var
  DirInfo  : SearchRec;
  Line     ,
  Position : integer;

begin
  LowVideo;
  GotoXY(1,FromLine); ClrEol;
  Line:=FromLine; Position:=1;
  FindFirst(MatchPtrn,$37,DirInfo);
  if DosError<>0 then
    writeln('*** NO FILES FOUND ***')
  else
  while (DosError=0) and (Line < ToLine ) do
  begin
    GotoXY(Position,Line);
    if DirInfo.Attr=$10 then HighVideo;
    write(DirInfo.Name);
    LowVideo;
    Position:=Position+16;
    if Position>65 then
    begin
      Line:=Line+1;
      Position:=1;
    end;
    FindNext(DirInfo);
  end;
  NormVideo;
end;



(*  This portion reprinted from "Turbo Pascal 4.0 unit to read a
    directory and display it" by Dipl. Phys. Christian Boettger, with
    the permission of Ir. Alexander Verbraeck, the original author.
********************** End Verbraeck Excerpt  ********************)
```

```
(*********************  Start Boettger Excerpt  *********************)

procedure ShowDir(MatchPtrn : string; FromLine, ToLine : integer;
                  var error : integer);
(*************************************************************************
       Christian Boettger                phone:  (+49) (0)531/391-5113
mail:  Institut fuer Metallphysik und Nukleare Festkoerperphysik,
       (room -167/-168), Technische Universitaet Braunschweig,
       Mendelssohnstrasse 3, D-3300 Braunschweig, land
       Bundesrepublik Deutschland (West Germany / FRG / RFA)
EARN:  I2010506@DBSTU1.BITNET       InterNet:   boettger@julian.uwo.CA
                                    UseNet:   boettger@julian.UUCP
UUCP / UseNet:
  (whereever)!uunet!watmath!julian!boettger
  (whereever)!uunet!boettger@hydra.uwo.CA
  (whereever)!uunet!mcvax!unido!i2010506@DBSTU1.BITNET
*************************************************************************)
   var DirInfo : SearchRec;
       start,i,
       line,ml : integer;

   procedure WriteEntry(DirInfo : SearchRec; line : integer);
      var DT       : DateTime;
          attribut : string;

      procedure GetAttribut (attr : byte; var attribut : string);
         begin
           case attr of
                ReadOnly  : attribut := 'ReadOnly';
                Hidden    : attribut := 'Hidden';
                SysFile   : attribut := 'SysFile';
                VolumeID  : attribut := 'VolumeID';
                Directory : attribut := 'Directory';
                Archive   : attribut := 'Archive';
                AnyFile   : attribut := 'AnyFile';
                else begin
                       Str(attr,attribut);
                       attribut := 'Attr = ' + attribut;
                     end;
                end;
         end;




(*  Reprinted from "Turbo Pascal 4.0 unit to read a directory and
    display it" by Dipl. Phys. Christian Boettger, by permission of the
    author.
********************** Continue Boettger Excerpt  *********************)
```

```
(********************* Continue Boettger Excerpt  ********************)
      begin  (*of WriteEntry*)
        with DirInfo do
           begin
             UnPackTime(Time,dt);
             GetAttribut(attr,attribut);
             GotoXY(1,line); ClrEol;
             IF attr = Directory THEN HighVideo;
             write ( Name );
             GoToXY ( 13, line );
             IF attr = Directory THEN
               Write ( '   <DIR>')
             ELSE Write ( size : 8 );
             GotoXy ( 24, line );
             {Write (Name:12,' ',Size:8,'   '); }
             with dt do
              begin
                write(day:2,'-',month:2,'-',year:4,'    ');
                write(hour:2,':',min:2,':',sec:2,'    ');
              end;
             writeln(' ',attribut);
             LowVideo;
           end;
      end;(*of WriteEntry*)

   begin (*of ShowDir*)
     M1 := ToLine - FromLine;
     start := WhereY+1;
     FindFirst(MatchPtrn, AnyFile, DirInfo);
     case DOSError of
          0 : begin
                WriteEntry(DirInfo,start);
                line := start;
                while DOSError=0 do
                 begin
                   FindNext(DirInfo);
                   Inc(line);
                   if line>M1 then begin
                     StandBy;
                     line := start;
                     ClrScr;
                   end;
                   if DosError=0 then WriteEntry(DirInfo,line)



(*  Reprinted from "Turbo Pascal 4.0 unit to read a directory and
    display it" by Dipl. Phys. Christian Boettger, by permission of thé
    author.
********************* Continue Boettger Excerpt  ********************)
```

```
(********************* Continue Boettger Excerpt  *********************)
                    else begin
                      GotoXY(1,line);
                      ClrEol;
                      writeln;
                      ClrEol;
                      writeln (Error_Code [ DOSError ],' !!');
                      writeln;
                      ClrEol;
                      GotoXY(1,WhereY);
                    end;
                  end;
                  error :=0;
              end;
           2 : begin
                GotoXY(1,start);
                writeln(Error_Code [ DOSError ],' !!');
                writeln('Directory not found!!');
                error := DOSError;
              end;
          18 : begin
                GotoXY(1,start);
                writeln(Error_Code [ DOSError ],' !!');
                writeln(
                  'No Entries in directory that match pattern !!');
                error := DOSERROR;
              end;
          else begin
                GotoXY(1,start);
                writeln(Error_Code [ DOSError ],' !!');
                error := DOSError;
              end;
          end;
    end;  (*of ShowDir*)

end.



(*  Reprinted from "Turbo Pascal 4.0 unit to read a directory and
    display it" by Dipl. Phys. Christian Boettger, by permission of the
    author.
********************* End Boettger Excerpt  *********************)
```

{

APPENDIX R

SOURCE LISTING FOR UNIT ERRORCOD

```
}
(*************************************************************************)
(****                        ERRORCOD.PAS                             ****)
(****  This unit maps MS-DOS error codes returned by the              ****)
(****  operating system to strings to give the operator a             ****)
(****  human readable response.                                       ****)
(****                                                                 ****)
(****  Reference:  MS-DOS Version 3 Programmer's Utility Pack         ****)
(****              MS-DOS Reference Guide Volume 1                    ****)
(****              1986, pp. 4.86-4.88, 4.254-4.257.                  ****)
(****                                                                 ****)
(****  Developed by Nelson Ard                                        ****)
(****                                                                 ****)
(****  Last modification Sep 89                                       ****)
(*************************************************************************)

UNIT ErrorCod;

INTERFACE

USES Dos;

CONST Error_Code : ARRAY [0..88] OF
      string[40] = ('No errors',
                    'Invalid function code',
                    'File not found',
                    'Path not found',
                    'No file handles left',
                    'Access denied',
                    'Invalid handle',
                    'Memory control blocks destroyed',
                    'Insufficient memory',
                    'Invalid memory block address',
                    'Invalid environment',
                    'Invalid format',
                    'Invalid access code',
                    'Invalid data',
                    'RESERVED error code',
                    'Invalid drive',
                    'Attempt to remove the current directory',
                    'Not same device',
                    'No more files',
                    'Disk is write-protected',
                    'Bad disk unit',
```

```
'Drive not ready',
'Invalid disk command',
'CRC error',
'Invalid length (disk operation)',
'Seek error',
'Not an MS-DOS disk',
'Sector not found',
'Out of paper',
'Write fault',
'Read fault',
'General failure',
'Sharing violation',
'Lock violation',
'Wrong disk',
'FCB unavailable',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'RESERVED error code',
'Network request not supported',
'Remote computer not listening',
'Duplicate name on network',
'Network name not found',
'Network busy',
'Network device no longer exists',
'Net BIOS command limit exceeded',
'Network adapter hardware error',
'Incorrect response from network',
'Unexpected network error',
'Incompatible remote adapt',
'Print queue full',
'Queue not full',
'Not enough space for print file',
'Network name was deleted',
'Access denied',
'Network device type incorrect',
'Network name not found',
'Network name limit exceeded',
'Net BIOS session time exceeded',
'Temporarily paused',
'Network request not accepted',
```

```
                    'Print or disk redirection is paused',
                    'RESERVED error code',
                    'RESERVED error code',
                    'RESERVED error code',
                    'RESERVED error code',
                    'RESERVED error code',
                    'RESERVED error code',
                    'RESERVED error code',
                    'File exits',
                    'Duplicate File Control Block',
                    'Cannot make',
                    'Interrupt 24 failure',
                    'Out of structures',
                    'Already assigned',
                    'Invalid password',
                    'Invalid parameter',
                    'Net write fault');

CONST Error_Class : ARRAY [1..12] OF string[40] =
                   ('Out of a resource',
                    'Temporary situation',
                    'Permission problem',
                    'Internal error in system software',
                    'Hardware failure',
                    'System software failure',
                    'Application program error',
                    'File or item not found',
                    'File or item of invalid format',
                    'File or item interlocked',
                    'Media failure - storage medium',
                    'Unknown error');

      Recommended_Error_Action : ARRAY [1..7] OF String[40] =
                   ('Retry, then prompt user',
                    'Retry after a pause',
                    'Reprompt user to reenter',
                    'Terminate with clean up',
                    'Terminate immediately',
                    'Observe only',
                    'Retry after correcting fault');

      Error_Locus : ARRAY [1..5] OF String[40] =
                   ('Unknown',
                    'Random Access block device',
                    'Related to a network',
                    'Related to serial access device',
                    'Related to RAM');
```

```
PROCEDURE Extended_Error_Code (VAR Error_Code  : INTEGER;
                               VAR Error_Class : Byte;
                               VAR Error_Locus : Byte);

{ Following an error code returned by an MS-DOS function call or
  I/O function, this may be called for amplification on the
  error }

IMPLEMENTATION

Var index : integer;

PROCEDURE Extended_Error_Code (VAR Error_Code  : INTEGER;
                               VAR Error_Class : Byte;
                               VAR Error_Locus : Byte);

Var Regs : Registers;

BEGIN
  Regs.AH := $59;
  Regs.BX := 0;
  Intr($21, Regs);
  Error_Code := Regs.AX;
  Error_Class := Regs.BH;
  Error_Locus := Regs.CH;
END;

BEGIN
END.
```

{

APPENDIX S

SOURCE LISTING FOR UNIT GENERAL

```
}
(*************************************************************************)
(**                          GENERAL.PAS                                **)
(** This is a library of general purpose routines to augment the       **)
(** features of Turbo Pascal 4.0.  This UNIT requires the standard     **)
(** units CRT and DOS supplied with the Turbo Pascal 4.0 compiler      **)
(** in order to compile.                                               **)
(**                                                                    **)
(** Reference:  Edwards, C. C., Advanced Techniques in                 **)
(**             Turbo Pascal, pp. 66 - 73, Sybex, Inc., 1987           **)
(**                                                                    **)
(** Modified from a Turbo Pascal 3.0 include file to a                 **)
(** Turbo Pascal 4.0 UNIT by Nelson Ard                               **)
(**                                                                    **)
(** Last Modification:  Sep 89                                         **)
(*************************************************************************)
(************************  Start Edwards Excerpt  *********************)


UNIT General;

INTERFACE

USES
     Dos,
     Crt;

TYPE
     Long_String = String[255];

     Hex_Type = String[2];

     Cursor_Type=(Cursor_Small,Cursor_Large,Cursor_Invisible);




(*  Reprinted with some modification from Advanced Techniques in Turbo
    Pascal by Charles Edwards, by permission of Sybex, Inc.  Copyright
    1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Procedure FillWord(Var V; Num,Value:Integer);
(*This procedure is similar to the Turbo procedure FillChar, except
  that it fills the variable with a 16 bit word value rather than an
  8 bit character.*)

Procedure Exchange(Var S,D; L:Integer);
(*This procedure is a fast machine languge routine to exchange the
  contents of two variables. No test is made concerning the
  compatibility of the variables.  That is left to the programmer.*)

Procedure Beep(Freq:Integer);
(*This procedure produces a tone for 1/4 second*)

Function Max(X,Y:Integer):Integer;
(*Max returns the larger of two integers*)

Function Min(X,Y:Integer):Integer;
(*Max returns the smaller of two integers*)

Procedure Cursor_Size(Size:Cursor_Type; Mono:Boolean);
(*This procedure changes the cursor into either an underline or a
  block cursor

 Input: Size = Cursor_Small creates an underline cursor
               Cursor_Large creates a block cursor
               Cursor_Invisible creates an invisible cursor

        Mono = True for a monochrome screen
               False for a color/graphics card
*)

Function Get_Time:Long_String;
(*This procedure returns the time in the form HH:MM:SS xM*)




(*  Reprinted with some modification from Advanced Techniques in Turbo
    Pascal by Charles Edwards, by permission of Sybex, Inc.  Copyright
    1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
IMPLEMENTATION

(********************  Continue Edwards Excerpt  ********************)

Procedure FillWord(Var V; Num,Value:Integer);
(*This procedure is similar to the Turbo procedure FillChar, except
  that it fills the variable with a 16 bit word value rather than an
  8 bit character.

Input: V: The variable which is to be filled
       Num: The number of words to full with the value
       Value: The 16 bit word to be stored in V
*)
     Begin
     Inline($C4/$BE/V                  (*LES DI,V[BP]*)
           /$8B/$8E/Num                (*MOV CX,[Num+BP]*)
           /$8B/$86/Value              (*MOV AX,[Value+BP]*)
           /$FC                        (*CLD*)
           /$F2/$AB                    (*REPNZ STOSW*)
           );
     End;  (*of FillWord*)

Procedure Exchange(Var S,D; L:Integer);
(*This procedure is a fast machine languge routine to exchange the
  contents of two variables. No test is made concerning the
  compatibility of the variables.  That is left to the programmer.

Input: S,D: The variables to be exchanged
       L: The number of bytes to exchange
*)
     Begin
     Inline($1E                        (*PUSH DS*)
           /$C5/$B6/S                  (*LDS SI,S[BP]*)
           /$C4/$BE/D                  (*LES DI,D[BP]*)
           /$8B/$8E/L                  (*MOV CX,[L+BP]*)
           /$FC                        (*CLD*)
           /$26/$8A/$05                (*L: MOV AL,ES:[DI]*)
           /$86/$04                    (*EXCH [SI],AL*)
           /$46                        (*INC SI*)
           /$AA                        (*STOSB*)
           /$E2/$F7                    (*LOOP L*)
           /$1F                        (*POP DS*)
           );
     End;  (*of Exchange*)


(*  Reprinted with some modification from Advanced Techniques in Turbo
    Pascal by Charles Edwards, by permission of Sybex, Inc.  Copyright
    1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
Procedure Beep(Freq:Integer);
(*This procedure produces a tone for 1/4 second*)
     Begin
     NoSound;  (*Reset flag*)
     Sound(Freq);
     Delay(250);
     Nosound;
     End;  (*of Beep*)

Function Max(X,Y:Integer):Integer;
(*Max returns the larger of two integers*)
     Begin
     If X < Y then
       Max:=Y
     else
       Max:=X;
     End;  (*of Max*)

Function Min(X,Y:Integer):Integer;
(*Max returns the smaller of two integers*)
     Begin
     If X < Y then
       Min:=X
     else
       Min:=Y;
     End;  (*of Min*)

Procedure Cursor_Size(Size:Cursor_Type; Mono:Boolean);
(*This procedure changes the cursor into either an underline or
  a block cursor

 Input: Size = Cursor_Small creates an underline cursor
               Cursor_Large creates a block cursor
               Cursor_Invisible creates an invisible cursor

        Mono = True for a monochrome screen
               False for a color/graphics card
*)
Const
  Cursor_Values:Array [0..3] of Integer = ($0607,$0007,$0C0D,$000D);
Var Regs:Registers;
     Begin
     Regs.AX:=$0100;
     If Size = Cursor_Invisible then
        Regs.CX:=$2607

(*  Reprinted with some modification from Advanced Techniques in Turbo
    Pascal by Charles Edwards, by permission of Sybex, Inc.  Copyright
    1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

     else
        Regs.CX:=Cursor_Values[Ord(Mono)*2+Ord(Size)];
     Intr($10,Regs);
     End;  (*of Cursor_Size*)

Function Get_Time:Long_String;
(*This procedure returns the time in the form HH:MM:SS xM*)
Var Regs:Registers;
    Hour,Min,Sec,M:String[2];
    I:Byte;
   Begin
   Regs.AH:=$2C;
   MSDos(Regs);
   Str(Regs.CL:2,Min);
   Str(Regs.DH:2,Sec);
   For I:=1 to 2 do
      Begin
      If Min[I]=' ' then Min[I]:='0';
      If Sec[I]=' ' then Sec[I]:='0';
      End;
   Case Regs.CH of
      0:  I:=12;
 13..23:  I:=Regs.CH-12;
   else   I:=Regs.CH;
      End;  (* of case*)
   Str(I:2,Hour);
   If Hour[1]=' ' then Hour[1]:='0';
   If Regs.CH < 12 then
      M:='AM'
   else
      M:='PM';
   Get_Time:=Hour+':'+Min+':'+Sec+' '+M;
   End;  (*of Get_Time*)

BEGIN
END.
```

(* Reprinted with some modification from Advanced Techniques in Turbo Pascal by Charles Edwards, by permission of Sybex, Inc. Copyright 1987 Sybex, Inc. All rights reserved.
********** Continue Edwards Excerpt **********)

{

APPENDIX T

SOURCE LISTING FOR UNIT MISCPACK

```
}
(*************************************************************************)
(****                        MISCPACK.PAS                             ****)
(****  This contains common data structure declarations for       ****)
(****  several units and a couple of utility routines.            ****)
(****  Derived from the include file of the same name in the      ****)
(****  reference.                                                 ****)
(****                                                             ****)
(****  Reference:  Swan, Turbo Pascal Files, 1987, pp. 14 - 26    ****)
(****  Developed by Nelson Ard                                    ****)
(****                                                             ****)
(****  Last modification Sep 89                                   ****)
(*************************************************************************)

UNIT Miscpack;

{ USES no other packages }
{ 15 Jul 89 - Added string128, response_type }
{ 19 Jul 89 - Added buffer for xmodm }
{ 11 Sep 89 - deleted Val2Hex }


INTERFACE

CONST

(************************  Start Swan Excerpt  ***************************)
{ String Lengths }

  PathLen = 65;  { Maximum complete path name length + 1 }
  FileLen = 12;  { Maximum file name length (with extension) }
  NameLen = 8;   { Maximum file name length (without extension) }
  ExtnLen = 3;   { Maximum file extension length }
  DriveLen = 2;  { Maximum drive letter string }

{ Typing helpers }

  NullStr = '';  { No blank between the quotes }
  Blank   = ' '; { A single blank character }

(*  Reprinted with some modification from Mastering Turbo Pascal Files
    By Tom Swan, by permission of Howard W. Sams and Company.  Copyright
    1987 Howard W. Sams and Company.  All rights reserved.
************************  Continue Swan Excerpt  ************************)
```

```
(************************  Continue Swan Excerpt  **********************)

{ Keyboard control code translations }

  KeyRight= ^D;     { Right arrow }
  KeyHome = ^W;     { Home        }
  KeyUp   = ^E;     { Up arrow    }
  KeyPgUp = ^R;     { PgUp        }
  KeyLeft = ^S;     { Left arrow  }
  KeyEnd  = ^Z;     { End         }
  KeyDown = ^X;     { Down arrow  }
  KeyPgDn = ^C;     { PgDn        }
  KeyIns  = ^V;     { Ins         }
  KeyDel  = ^G;     { Del         }


TYPE

  { File and path name strings }

  PathString  = String[ PathLen  ];  { C:\TURBO\TEST.PAS }
  FileString  = String[ FileLen  ];  { TEST.PAS          }
  NameString  = String[ NameLen  ];  { TEST              }
  ExtnString  = String[ ExtnLen  ];  { PAS               }
  DriveString = String[ DriveLen ];  { C:                }

{ Other strings }

  HexStr      = String[ 4 ];         { 4 - digit hex strings (FC9A) }
  Str80       = String[ 80 ];        { 80-character strings }
  string128   = String[ 128 ];

{Miscellaneous types }

  Pointer     = ^Byte;               { Pointer to memory bytes }
  CharSet     = SET OF CHAR;         { Character sets }


{ Added for Spawn, Intrinsic Exec calls }

  Response_type = ( strng, file_type, nothing );

{ Added for Parser, xmodm }

CONST
     Maxblock = 1024;

(*  Reprinted with some modification from Mastering Turbo Pascal Files
    By Tom Swan, by permission of Howard W. Sams and Company.  Copyright
    1987 Howard W. Sams and Company.  All rights reserved.
************************  Continue Swan Excerpt  **********************)
```

```
(************************  Continue Swan Excerpt  **********************)

TYPE
  Buffer = ARRAY [1..Maxblock] OF CHAR;

PROCEDURE BumpStrUp ( VAR s : String );

{ Convert (bump) all chars in string s to uppercase }

IMPLEMENTATION

PROCEDURE BumpStrUp (VAR s : String);

{ Convert (bump) all chars in string s to uppercase }

VAR

  i : INTEGER;  {String index }

BEGIN
  FOR i := 1 to Length( s ) DO
    S[i] := UpCase( s[i] )
END; { BumpStrUp }

BEGIN  {Unit initializaton }

END.   { UNIT Miscpack }
```

```
(*  Reprinted with some modification from Mastering Turbo Pascal Files
    By Tom Swan, by permission of Howard W. Sams and Company.  Copyright
    1987 Howard W. Sams and Company.  All rights reserved.
**************************  End Swan Excerpt  **********************)
```

{

APPENDIX U

SOURCE LISTING FOR UNIT PARSER

```
}
(*************************************************************************)
(****                        PARSER.PAS                               ****)
(****  This is the unit that executes all commands for the        ****)
(****  Slave computer.                                            ****)
(****                                                             ****)
(****                                                             ****)
(****  References:  Hall, W.V., "When Turbo Isn't Enough," in     ****)
(****               Shammas, N.C., The Turbo Pascal Toolbook,     ****)
(****               pp. 145 - 146, M & T Publishing, Inc., 1986.****)
(****                                                             ****)
(****               Mefford, M.J., "Running Programs Painlessly ****)
(****               PC Magazine, v. 7, 16 February, 1988.         ****)
(****                                                             ****)
(****  Developed by Nelson Ard                                    ****)
(****                                                             ****)
(****  Last modification Sep 89                                   ****)
(*************************************************************************)

UNIT Parser;

{ 8 Nov 88 }
{ 5 June 89  - changed sets to constants}
{ 9 June 89 -added
             argv, argc functions
             adjusted parsename to correctly parse long filenames}
{ 19 Jun 89 - added buf_to_string, string_to_buf}
{ 20 Jun 89 - added Resolve_command to prepare for EXEC call }
{ 12 Jul 89 - moved Match_command, internal command constructs to spawn,
              added response construct to parser_main }
{  4 Aug 89 - deleted Intrinsic from USES statement  }

INTERFACE

USES MISCPACK, ErrorCod, Spawn, Dos;

PROCEDURE Parser_main (     Command_s : string128;
                        VAR Response  : String128;
                        VAR Restype   : Response_type;
                        VAR Error_msg : String128;
                        VAR Errtype   : Response_type;
                        VAR Prompt    : String128 );
```

```
{ This procedure parses an MS-DOS command and executes it locally

  Input:  Command_s is the command to be executed with path

  Output: Response is the output of the program
          Restype is the type of Response (string, file, nothing)
          Error_Msg is the error output of the program
          Errtype is the type of Error_Msg (string, file, nothing)
          Prompt is a simulated command line prompt after program
          execution

}

IMPLEMENTATION

TYPE
  argtype = ( opt, other );

  arg_rec  = RECORD
               arg_type     : argtype;
               arg_length   : byte;
               arg_index    : byte;
             END;

  argarray = ARRAY [0..9] OF arg_rec;

  SETOFCHAR = set of char;

  command_buffer = ARRAY [1..12] OF char;

CONST
  SPACE     = ' ';
  TAB       = ^H;
  COMMA     = ',';
  SEMICOLON = ';';
  COLON     = ':';
  PLUS      = '+';
  MINUS     = '-';
  SLASH     = '/';
  BACKSLASH = '\';
  DOT       = '.';
  STAR      = '*';
  NUL       = ^@;
  TILDE     = '~';

  Path_or_drive : SETOFCHAR = [ COLON, BACKSLASH ];
  Node_or_drive : SETOFCHAR = [ COLON ];
  arg_separator : SETOFCHAR = [ SPACE, COMMA, SEMICOLON, PLUS, MINUS ];
  whitespace    : SETOFCHAR = [ SPACE, TAB ];
  option        : SETOFCHAR = [ SLASH ];
  NullString : PathString = '';
```

```
VAR
  arg_array : argarray;
  arg_count : byte;
  Command_line : PathString;
  index : byte ;
  count : byte;
  (* This variable for use ** only ** by argv() *)

  arg_string : string128;

PROCEDURE Parse ( Command : string128 );

{ Used by Parser_Main to count and isolate the command line
  parameters.  This procedure loads argc and argv }

VAR
  index : byte;

BEGIN
  arg_string := Command; {save a copy of the command}
  FOR arg_count := 0 TO 9 DO
  WITH Arg_array [ arg_count ] DO BEGIN
    arg_type := OTHER;
    arg_length := 0;
    arg_index := 0;
  END;
  index := 1;
  arg_count := 0;
  REPEAT
    WHILE (index < Length ( Command ) )
      AND ( Char ( Command [ index ]) IN whitespace ) DO
      INC(index);
    WITH arg_array [ arg_count ] DO BEGIN
  IF index <= Length ( Command ) THEN
    CASE Command [ index ] OF
      TAB, SPACE : BEGIN
                   END;

      SLASH      : IF index < length ( Command ) THEN
                     BEGIN
                       (*INC ( index );*)
                       arg_length := 2;  (*!*)
                       arg_index  := index;
                       arg_type   := opt;
                       INC ( index );
                       INC ( index );
                       INC ( arg_count );
                     END;
```

```
      ELSE          BEGIN
                      arg_index := index;
                      arg_type  := other;
                      arg_length := 1;
                          INC ( index );
                      WHILE ( index <= Length ( Command ) ) AND
                        NOT ( Char (Command [ index ]) IN whitespace )
                        AND NOT ( Char (Command [ index ]) IN option )
                        DO BEGIN
                          INC ( arg_length );
                          INC ( index );
                        END;
                      INC ( arg_count );
                    END; {BEGIN}
      END {CASE}
    END { WITH }
  UNTIL index >= length ( Command );
END; {Parse}

FUNCTION argc : byte;

{ Returns a count of the number of arguements on the command
  line }

BEGIN
  argc := arg_count;
END;

FUNCTION argv ( arg_count : byte ) : string128;

{ Returns the arg_count'th arguement from the command
  line }

VAR
  index : byte;
  temp  : string128;

BEGIN
  temp := Nullstring;
  WITH arg_array [ arg_count ] DO
    FOR index  := arg_index TO (arg_index + arg_length - 1) DO
      temp := temp + arg_string [ index ];
  argv := temp;
END;
```

```
(***************************  Start Hall Excerpt  ********************)

PROCEDURE ParseName ( inName : PathString; VAR nameSpec  : NameString;
                                           VAR extnSpec  : ExtnString;
                                           VAR fylespec  : Filestring;
                                           VAR pathSpec  : PathString;
                                           VAR driveSpec : DriveString;
                                           VAR nodeSpec  : NameString);

{ Breaks down a filespec into its component parts for Parser_Main,
  Resolve_command.  From the Hall reference.  }

VAR
  Count : Byte;
  DotPos : Byte;
  StarPos : Byte;
  index : integer;
  filespec : pathstring;

BEGIN
  Count := Length ( InName);
  {$V-}
    MiscPack.BumpStrUp( InName );
  {$V+}
  IF (InName[Count] IN Path_or_drive) THEN
    { do nothing }
  ELSE BEGIN
    REPEAT
      Count := PRED(Count);
    UNTIL (Count = 0) OR (InName[Count] IN Path_or_drive) ;
  END;


{Isolate Filename}
    {           Copy (Source, Startpos , No of Char)  }
    fileSpec := Copy (InName, Count + 1, LENGTH (InName) - Count);
    DELETE (InName, Count + 1, LENGTH(InName) - Count); {Chop tail off}
    IF (Count > 2) THEN
      IF (InName[Count] <> ':')  THEN
      REPEAT
        Count := PRED(Count);
      UNTIL (InName[Count] IN Node_or_Drive) OR (Count = 0);



(*  The library ParseName appears in The Turbo Pascal Toolbook by Namir
    C. Shammas (ed.) and has been reprinted with the permission of the
    publisher M & T Books 1-800-533-4372.  Minor modifications by Nelson
    Ard.
*************************  Continue Hall Excerpt  ********************)
```

```
(************************  Continue Hall Excerpt  **********************)

    CASE Count OF

      0 : pathSpec := InName;

      1 : { Syntax Error };

      ELSE BEGIN
          pathSpec := Copy (InName, Count + 1, LENGTH (InName) - Count);
          {Chop tail off}
          DELETE (InName, Count + 1, LENGTH(InName) - Count);

          CASE InName[PRED(Count)] OF

          COLON : BEGIN
                    {Chop tail off}
                    DELETE (InName, Length (InName) - 1, 2);
                    nodeSpec := InName;
                  END;

        'a'..'z',
        'A'..'Z'  : BEGIN
                      driveSpec := InName[PRED(Count)] + ':';
                      DELETE (InName, Count - 1, 2); {Chop tail off}
                      Count := Length(InName);
                      IF (Count > 2) AND (POS('::', InName) = Count - 1)
                      THEN IF LENGTH(InName) > 10 THEN
                             nodeSpec := Copy (InName, 1, 8)
                           ELSE nodeSpec := Copy (InName, 1,
                                                  LENGTH (InName) - 2)
                      ELSE { Syntax error in node part  }
                      IF Count <> 0 THEN ;
                    END;

          ELSE { Syntax Error, drive not alpha character };

          END; {Case}
        END;
    END; {Case}

{Adjust filename}
    DotPos := POS( DOT, fileSpec);
    IF DotPos <> 0 THEN BEGIN
      extnSpec := COPY(fileSpec, DotPos + 1, 3);

(*  The library ParseName appears in The Turbo Pascal Toolbook by Namir
    C. Shammas (ed.) and has been reprinted with the permission of the
    publisher M & T Books 1-800-533-4372.  Minor modifications by Nelson
    Ard.
************************  Continue Hall Excerpt  **********************)
```

```
(************************  Continue Hall Excerpt  **********************)

      DELETE(fileSpec, DotPos, (LENGTH(fileSpec) - DotPos)+1);
    END
    ELSE
      extnSpec := '';
   IF LENGTH(fileSpec) > 8 THEN
      DELETE(fileSpec, 9, LENGTH(fileSpec)-8);
    StarPos := POS( STAR, fileSpec );
   IF StarPos <> 0 THEN BEGIN
        DELETE(fileSpec, StarPos, (LENGTH(fileSpec)-StarPos)+1);
        FOR Count := LENGTH(fileSpec) TO 7 DO
          fileSpec:= fileSpec + '?';
        END;
   Namespec := filespec;
   StarPos := POS (STAR, extnSpec);
   IF StarPos <> 0 THEN BEGIN
     DELETE(extnSpec, StarPos, (LENGTH(fileSpec)-StarPos)+1);
     FOR Count := LENGTH(extnSpec) TO 2 DO
       extnSpec := extnSpec + '?';
     END;
   IF NOT ( extnspec = Nullstring ) THEN
     fylespec := Namespec + DOT + extnspec
   ELSE fylespec := Namespec;
END;

(*  The library ParseName appears in The Turbo Pascal Toolbook by Namir
    C. Shammas (ed.) and has been reprinted with the permission of the
    publisher M & T Books 1-800-533-4372.  Minor modifications by Nelson
    Ard.
*****************************  End Hall Excerpt  **********************)

TYPE { used by Resolve_Command and Parser_Main }

  Type_of_file = (BAT_File, COM_File, EXE_File, Directoree, Other_File,
                  Pathname );

VAR { initialized by Parser_Main for resolve_command }
  pathSpec : PathString;
  fileSpec : fileString;
  nodeSpec,
  nameSpec : NameString;
  extnSpec : ExtnString;
  driveSpec : DriveString;
```

```
FUNCTION Resolve_command ( VAR arguement : PathString ) : Type_of_file;

{ The MS-DOS Exec function needs a complete file specification (drive,
  path and filename including extension to run a child process.
  Resolve_command examines the first arguement in an MS-DOS command
  line, arguement, and fills out the complete path information if
  needed, then uses this path to conduct a file search for the
  exact filename.  The completed file specification is returned to
  the caller along with the type (COM, EXE, BAT, or path) for
  execution or directory change action.  The building blocks
  needed to construct the complete file specification have been
  placed in the variable immediately above by ParseName.  The
  deterministic algorithm for detecting the correct executable file is
  from (Mefford, 1988, p. 327).

 Input:  arguement, the command file to be searched for

 Output: arguement, adjusted to specify a complete path
         The function returns the type of file as an enumerated type

}

VAR
  DirInfo : SearchRec;
  resolved,
  relative_directory : boolean;
  Dir : PathString;

BEGIN
  resolved := FALSE;
  GetDir ( 0, Dir );
  { lack of a leading backslash could mean a simple
    request to log to another drive }
  relative_directory := ( pathSpec [ 1 ] <> BACKSLASH );
  IF relative_directory AND (( Dir [1] = driveSpec [1] )
                        OR (Drivespec = BLANK )) THEN
    { Fill out the complete path specification }
    arguement := Dir + BACKSLASH + arguement;

  IF extnSpec = NullString THEN BEGIN

  { The command does not have a file extension, could be a
    directory.  Search the now complete path for a file
    with the same name, in the reverse order that the
    MS-DOS command processor would.  Add the appropriate
    extension to arguement if matched.  End up with the
    file with precedence to execute.  }
```

```
  FindFirst ( arguement + '.BAT', Archive, DirInfo);
  WHILE DosError = 0 DO
    BEGIN
      IF DirInfo.attr AND Archive <> 0 THEN BEGIN
        arguement := arguement + '.BAT';
        resolve_command := BAT_File;
        resolved := TRUE;
      END;
    FindNext (DirInfo);
    END;

  FindFirst ( arguement + '.COM', Archive, DirInfo);
  WHILE DosError = 0 DO
    BEGIN
      IF DirInfo.attr AND Archive <> 0 THEN BEGIN
        arguement := arguement + '.COM';
        resolve_command := COM_File;
        resolved := TRUE;
      END;
    FindNext (DirInfo);
    END;

  FindFirst ( arguement + '.EXE', Archive, DirInfo);
  WHILE DosError = 0 DO
    BEGIN
      IF DirInfo.attr AND Archive <> 0 THEN BEGIN
        arguement := arguement + '.EXE';
        resolve_command := EXE_File;
        resolved := TRUE;
      END;
    FindNext (DirInfo);
    END;
  END
ELSE BEGIN  { extension not NULL, ready to execute }
  IF (extnSpec = 'COM') THEN BEGIN
    Resolve_command := COM_File;
    resolved := TRUE;
  END
  ELSE IF (extnSpec = 'BAT') THEN BEGIN
    Resolve_command := BAT_File;
    resolved := TRUE;
  END
  ELSE IF (extnSpec = 'EXE') THEN BEGIN
    Resolve_command := EXE_File;
    resolved := TRUE;
  END
  ELSE BEGIN
    Resolve_command := Other_file; { a path specification ? }
    resolved := TRUE;
  END
END;
```

```
(* changed this *)
IF NOT resolved THEN BEGIN
  FindFirst ( arguement , Directory, DirInfo);
  WHILE DosError = 0 DO
    BEGIN
      IF DirInfo.attr AND Directory <> 0 THEN BEGIN
        Resolve_command := Directoree;
        resolved := TRUE;
      END;
    FindNext (DirInfo);
    END;
END;

  IF NOT resolved THEN Resolve_command := Pathname;
END; {Resolve_Command}


(*-----------------------------------------------------------------*)

PROCEDURE Parser_main (     Command_s : string128;
                        VAR Response  : String128;
                        VAR Restype   : Response_type;
                        VAR Error_msg : String128;
                        VAR Errtype   : Response_type;
                        VAR Prompt    : String128 );


{ This procedure parses a command line similar in form to an
  MS-DOS command, and executes it if possible on the local
  machine

  Input:  Command_s is the command to be executed with path

  Output: Response is the output of the program
          Restype is the type of Response (string, file, nothing)
          Error_Msg is the error output of the program
          Errtype is the type of Error_Msg (string, file, nothing)
          Prompt is a simulated command line prompt after program
          execution

}

CONST NullString : String = '';
      Current_Drive : byte = 0; { used with ChDir }


VAR
  Command : Internal_Command;
  arg_count : byte;
  index : byte;
  cmdline,
```

```
  Current_Dir,
  program_name : PathString;
  File_type : Type_of_File;
  Batch : boolean;

  PROCEDURE Init_parse;

  { Break the command line into parameters, store the components
    of the first arguement (normally the command itself) }

  VAR
    index : byte;

  BEGIN
    Parse (command_s); { load argc, argv }
    pathspec  := argv(0);
    filespec  := Nullstring;
    extnspec  := Nullstring;
    drivespec := NullString; {Blank; !!}
    nodespec  := Nullstring;
    { now break the first arguement into components }
    Parsename ( pathspec, NameSpec, extnSpec, fileSpec,
                pathspec, drivespec, nodeSpec );

  END; { Init_Parse }

BEGIN
  Init_Parse;
  IF  ( Length ( Drivespec ) = 2 ) AND ( argc = 1 )
    { Drive change only }
    THEN BEGIN
      command_s := 'CD ' + command_s;
      Init_Parse; { redo with added command }
    END;

  IF Match_command ( FileSpec, Command ) THEN BEGIN
    { command can be handled by * this * program }
    IF argc >= 1 THEN BEGIN
      cmdline := Nullstring; { no command tail }
      FOR index := 1 TO (argc - 1) DO
        Cmdline := Cmdline + argv(index) + SPACE;
      { trim trailing space }
      IF Cmdline [ Length ( Cmdline ) ] = SPACE THEN
        Cmdline := Copy ( Cmdline, 1, Length ( Cmdline ) - 1 );

    END;
    { process as a built in function }
    Process_intrinsic_command ( Command, cmdline, Response, Restype,
                                Error_msg, Errtype, Prompt );
  END
```

```
  ELSE BEGIN { prepare for a child process }
    program_name := argv(0);
    File_Type := Resolve_command ( Program_Name );
    CASE File_Type OF
      COM_File,
      EXE_File,
      BAT_File : BEGIN
                   Batch := ( File_Type = BAT_File );
                   cmdline := NullString;
                   IF argc > 1 THEN FOR index := 1 TO argc - 1 DO
                     Cmdline := Cmdline + SPACE + argv(index);
                   Run_Local ( Program_name, cmdline, Response, Restype,
                               Error_msg, Errtype, Prompt, Batch  );
                 end;
      ELSE BEGIN { command did not parse, notify Master }
             Errtype := nothing;
             System.GetDir ( Current_Drive, Prompt );
             Prompt := Prompt + '>';
             Restype := strng;
             Response := 'Slave:  syntax error';
           END; {ELSE}
    END; {CASE}
  END;
END; {Parser main}

BEGIN

END.
```

{

APPENDIX V

SOURCE LISTING FOR UNIT REDIRECT

```
}
(*************************************************************************)
(****                        REDIRECT.PAS                             ****)
(****  This is the unit that accomplishes redirection of the      ****)
(****  Standard Input and Output file handles normally assigned ****)
(****  by the MS-DOS command processor to files to capture the  ****)
(****  output of a program running under the Slave computer     ****)
(****  control.  Variables are loaded with the file names for   ****)
(****  later reference.                                         ****)
(****                                                           ****)
(****  Reference:  Defenbaugh, G., "Parents, Children,          ****)
(****              Redirection, and Piping with DOS Functions   ****)
(****              45H and 46H, Programmer's Journal, Nov/Dec   ****)
(****              1986, pp. 22-25.                             ****)
(****                                                           ****)
(****  Developed by Nelson Ard                                  ****)
(****                                                           ****)
(****  Last modification Sep 89                                 ****)
(*************************************************************************)

UNIT Redirect;

(* Modification history

   22 Jul 89 - Chained ErrorNum variables through Close_File_Handle
               call
             - Placed two string variables in interface section for
               external units to find filespec for the response, error
               files while using standard TP file functions
    4 Aug 89 - Absorbed FileDecl UNIT as include file *)


INTERFACE

USES Dos, Crt, Miscpack;

PROCEDURE Restore_CRT_Assignments;

{ Optional procedure to replace the standard files Input and Output
  with textfile drivers in the CRT unit for speed.  In turns out that
  the CRT Unit does this on initialization, but disallows I/O
  redirection by doing so
  (Turbo Pascal Owner's Handbook, 1987, p. 377) }
```

```
PROCEDURE Init_Redirect_Unit;

{ Required to reset I/O to the MS-DOS standard file handles, which
  may then be redirected }

FUNCTION Redirect_Std_Input  : boolean;

{ Redirect program input from a predefined file }

FUNCTION Redirect_Std_Output : boolean;

{ Redirect program output to a predefined file }

FUNCTION Redirect_Std_Error    : boolean;

{ Redirect program error output to a predefined file }

FUNCTION Redirect_All_Output : boolean;

{ Redirect program output and error output to a predefined file }

FUNCTION Restore_Std_Input  : boolean;

{ Restore program input to the standard file handle }

FUNCTION Restore_Std_Output : boolean;

{ Restore program output to the standard file handle }

FUNCTION Restore_Std_Error    : boolean;

{ Restore program error output to the standard file handle }

FUNCTION Restore_All_Output    : boolean;

{ Restore program output and error output to the standard file handle }

VAR
  Response_File,
  Errors_File      : PathString;

IMPLEMENTATION

CONST {These are the predefined standard and redirected files}
      {MS-DOS predefines the following handles}
  StdIn  : word = 0;  (* File handle for Standard Input *)
  StdOut : word = 1;  (* File handle for Standard Output *)
  StdErr : word = 2;  (* File handle for Standard Error *)
  StdAux : word = 3;  (* File handle for Standard Auxiliary *)
  StdPrn : word = 4;  (* File handle for Standard Printer *)
      {Redirection takes place from/to these files}
```

```
  Std_Output_File_Temp : String[21] = 'C:\Scratch\OTPT.TMP';
  Std_Input_File_Temp :  String[21] = 'C:\Scratch\INPT.TMP';
  Std_Error_File_Temp :  String[21] = 'C:\Scratch\Err.TMP';


CONST Make_Dir         : Byte = $39;
      Remove_Dir       : Byte = $3A;
      Change_Dir       : Byte = $3B;
      Create_Handle    : Byte = $3C;
      Open_Handle      : Byte = $3D;
      Close_Handle     : Byte = $3E;
      Read_Handle      : Byte = $3F;
      Write_Handle     : Byte = $40;
      Delete_Entry     : Byte = $41;
      Move_Ptr         : Byte = $42;
      Change_Mode      : Byte = $43;
      Dup_Handle       : Byte = $45;
      FDup_Handle      : Byte = $46;
      Get_Dir          : Byte = $47;
      Find_First_File  : Byte = $4E;
      Find_Next_File   : Byte = $4F;


VAR
  Input_File,
  Error_File,
  Output_File : Text;
  Saved_Std_In,
  Saved_Std_Out,
  Saved_Std_Err,
  RedirIn,
  RedirOut,
  RedirErr : word;

PROCEDURE Init_Redirect_Unit;

{ Optional procedure to replace the standard files Input and Output
  with textfile drivers in the CRT unit for speed.  In turns out that
  the CRT Unit does this on initialization, but disallows I/O
  redirection by doing so
  (Turbo Pascal Owner's Handbook, 1987, p. 377) }

BEGIN
  Assign  ( Input, '' );
  Reset   ( Input );
  Assign  ( Output, '' );
  Rewrite ( Output );
END;
```

```
FUNCTION Duplicate_Handle ( Handle : word;
                            VAR ErrorNum : word  ) : word;

{ Input:   Handle, a file handle to an open file
  Output:  The function returns a second file handle
           for the same file.  Both handles use the same
           file pointer
           ErrorNum is returned by MS-DOS:
             $04 : No free handles left
             $06 : Handle is not currently open

}

VAR Regs : Registers;

BEGIN
  Regs.AH := Dup_Handle;
  Regs.BX := Handle;
  Intr($21, Regs);
  IF ( Regs.Flags AND FCarry ) = 0 THEN BEGIN
    Duplicate_Handle := Regs.AX
  END
  ELSE BEGIN
    ErrorNum := ErrorNum + Regs.AX;
    Duplicate_Handle := $FF
  END
END;

FUNCTION Close_File_Handle ( Handle : word;
                             VAR ErrorNum : word ) : Boolean;

{ Input:   Handle, a file handle to an open file
  Output:  The function returns TRUE if the operation was successful
           and the file closed.  All internal buffers are flushed.
           If FALSE, an invalid handle was specified.
           ErrorNum is returned by MS-DOS:
             $06 : Handle is not currently open

}

VAR Regs : Registers;

BEGIN
  Regs.AH := Close_Handle;
  Regs.AL := $0;
  Regs.BX := Handle;
  Intr($21, Regs);
  IF ( Regs.Flags AND FCarry ) = 0 THEN BEGIN
    Close_File_Handle := TRUE
  END
```

```
  ELSE BEGIN
    ErrorNum := ErrorNum + Regs.AX;
    Close_File_Handle := FALSE;
  END
END;

PROCEDURE Redirect_Handle (     Handle, Red_Handle : word;
                            VAR ErrorNum : word  );

{ Input:   Handle, a file handle to an open file
           Red_Handle a file handle to a second file
  Output:  The file referenced by Red_Handle is closed
           Red_Handle now uses the same file pointer as
           Handle, and either may be used to acces the file
           ErrorNum is returned by MS-DOS:
             $04 : No free handles left
             $06 : Handle is not currently open
}

VAR Regs : Registers;

BEGIN
  Regs.AH := FDup_Handle;
  Regs.BX := Handle;
  Regs.CX := Red_Handle;
  Intr($21, Regs);
  IF ( Regs.Flags AND FCarry ) = 0 THEN BEGIN

  END
  ELSE BEGIN
    ErrorNum := ErrorNum + Regs.AX;
  END
END;

FUNCTION Redirect_Std_Output : boolean;

{ Redirect program output to a predefined file

  On entry, StdOut refers to the standard output
  device driver.  A copy of StdOut is saved, and
  StdOut is redirected to our predefined output file

  The function returns TRUE if successful
}

VAR ErrorNum : word;

BEGIN
  ErrorNum := 0;
  Assign ( Output_File, Std_Output_File_Temp );
  Rewrite ( OutPut_File );
```

```
  Saved_Std_Out := Duplicate_Handle ( StdOut, ErrorNum );
  Redirect_Handle ( TextRec( Output_File ).Handle, StdOut, ErrorNum );
  Redirect_Std_Output := ( ErrorNum = 0 );
END;

FUNCTION Restore_Std_Output : boolean;

{ Restore program output to the standard file handle

  On entry, StdOut refers to our predefined file
  StdOut is rereferenced to the standard output
  device driver

  The function returns TRUE if successful
}

VAR ErrorNum : word;

BEGIN
  ErrorNum := 0;
  Redirect_Handle ( Saved_Std_Out, StdOut, ErrorNum );
  IF Close_File_Handle ( Saved_Std_Out, ErrorNum ) THEN
  {$I-}
  Close ( Output_File );
  IF IOResult = 0 THEN BEGIN
    Response_File := Std_Output_File_Temp;
    Restore_Std_Output := ( ErrorNum = 0 );
    END
  ELSE BEGIN
    Response_File := NullStr;
    Restore_Std_Output := FALSE;
  END;
  {$I+}
END;

FUNCTION Redirect_Std_Input : boolean;

{ Redirect program input from a predefined file

  On entry, StdIn refers to the standard input
  device driver.  A copy of StdIn is saved, and
  StdIn is redirected to our predefined input file

  The function returns TRUE if successful
}

VAR ErrorNum : word;

BEGIN
  ErrorNum := 0;
  Assign ( Input_File, Std_Input_File_Temp );
```

```
  Reset  ( InPut_File );
  Saved_Std_In := Duplicate_Handle ( StdIn, ErrorNum );
  Redirect_Handle ( TextRec( Input_File ).Handle, StdIn, ErrorNum );
  Redirect_Std_Input := ( ErrorNum = 0 );
END;

FUNCTION Restore_Std_Input : boolean;

{ Restore program input to the standard file handle

  On entry, StdIn refers to our predefined file
  StdIn is rereferenced to the input
  device driver

  The function returns TRUE if successful
}


VAR ErrorNum : word;

BEGIN
  ErrorNum := 0;
  Redirect_Handle ( Saved_Std_In, StdIn, ErrorNum );
  {$I-}
  Close ( Input_File );
  IF Close_File_Handle ( Saved_Std_In, ErrorNum ) THEN;
  Restore_Std_Input := ( ErrorNum = 0 ) AND (IOResult <> 0);
  {$I+}
END;

FUNCTION Redirect_Std_Error : boolean;

{ Redirect program error output to a predefined file

  On entry, StdErr refers to the standard output
  device driver.  A copy of StdErr is saved, and
  StdErr is redirected to our predefined error file
  Overcomes inability to redirect from the MS-DOS
  command line

  The function returns TRUE if successful

}

VAR ErrorNum : word;

BEGIN
  ErrorNum := 0;
  Assign ( Error_File, Std_Error_File_Temp );
  Rewrite ( Error_File );
  Saved_Std_Err := Duplicate_Handle ( StdErr, ErrorNum );
```

```
  Redirect_Handle ( TextRec( Error_File ).Handle, StdErr, ErrorNum );
  Redirect_Std_Error := ( ErrorNum = 0 );
END;

FUNCTION Restore_Std_Error : boolean;

{ Restore program error output to the standard file handle

  On entry, StdErr refers to our predefined file
  StdErr is rereferenced to the output
  device driver

  The function returns TRUE if successful
}


VAR ErrorNum : word;

BEGIN
  ErrorNum := 0;
  Redirect_Handle ( Saved_Std_Err, StdErr, ErrorNum );
  {$I-}
  Close ( Error_File );
  IF Close_File_Handle ( Saved_Std_Err, ErrorNum ) THEN;
  IF IOResult = 0 THEN BEGIN
    Errors_File := Std_Error_File_Temp;
    Restore_Std_Error := ( ErrorNum = 0 );
    END
  ELSE BEGIN
    Errors_File := NullStr;
    Restore_Std_Error := FALSE;
  END;
  {$I+}

END;

FUNCTION Redirect_All_Output : boolean;

{ Redirect program output and error output to a predefined file

  On entry, StdOut refers to the standard output
  device driver.  A copy of StdOut is saved, and
  StdOut is redirected to our predefined output file

  On entry, StdErr refers to the standard output
  device driver.  A copy of StdErr is saved, and
  StdErr is redirected to our predefined error file
  Overcomes inability to redirect from the MS-DOS
  command line

  The function returns TRUE if successful
```

```
}

VAR ErrorNum : word;

BEGIN
  ErrorNum := 0;
  {$I-}
  Assign ( Output_File, Std_Output_File_Temp );
  Rewrite ( OutPut_File );
  Saved_Std_Out := Duplicate_Handle ( StdOut, ErrorNum );
  Saved_Std_Err := Duplicate_Handle ( StdErr, ErrorNum );
  Redirect_Handle ( TextRec( Output_File ).Handle, StdOut, ErrorNum );
  Redirect_Handle ( TextRec( Output_File ).Handle, StdErr, ErrorNum );
  Redirect_All_Output := ( ErrorNum = 0 ) AND (IOResult <> 0 );
  {$I+}
END;

FUNCTION Restore_All_Output : boolean;

{ Restore program output and error output to the standard file handle

  On entry, StdOut refers to our predefined file
  StdOut is rereferenced to the standard output
  device driver

  On entry, StdErr refers to our predefined file
  StdErr is rereferenced to the output
  device driver

  The function returns TRUE if successful

}
VAR
  ErrorNum : word;

BEGIN
  ErrorNum := 0;
  Redirect_Handle ( Saved_Std_Out, StdOut, ErrorNum );
  IF Close_File_Handle ( Saved_Std_Out, ErrorNum ) THEN;
  Redirect_Handle ( Saved_Std_Err, StdErr, ErrorNum );
  IF Close_File_Handle ( Saved_Std_Err, ErrorNum ) THEN;
  {$I-}
  Close ( Output_File );
  IF IOResult = 0 THEN BEGIN
    Response_File := Std_Output_File_Temp;
    Restore_All_Output := ( ErrorNum = 0 );
    END
  ELSE BEGIN
    Response_File := NullStr;
    Restore_All_Output := FALSE;
```

```
  END;
  {$I+}
END;

PROCEDURE Restore_CRT_Assignments;

{ Optional procedure to replace the standard files Input and Output
  with textfile drivers in the CRT unit for speed.  In turns out that
  the CRT Unit does this on initialization, but disallows I/O
  redirection by doing so
  (Turbo Pascal Owner's Handbook, 1987, p. 377)

}

BEGIN
  AssignCRT ( Input );
  Reset     ( Input );
  AssignCRT ( Output );
  Rewrite   ( Output );
END;

BEGIN (* no initialization required *)
END.
```

{

APPENDIX W

SOURCE LISTING FOR UNIT SPAWN

```
}
(*************************************************************************)
(****                        SPAWN.PAS                                ****)
(****  This is the unit that executes child processes under       ****)
(****  MS-DOS for the Slave computer.  Included is a function     ****)
(****  to detect MS-DOS commands to be handled by the             ****)
(****  program rather than by a spawned copy of Command.com.      ****)
(****  The function is placed here to prevent circular unit       ****)
(****  dependencies while restricting visibility to unrelated     ****)
(****  units.                                                     ****)
(****                                                             ****)
(****  Reference:  Mefford, M.J., "Running Programs Painlessly    ****)
(****              PC Magazine, v. 7, 16 February, 1988.          ****)
(****                                                             ****)
(****  Developed by Nelson Ard                                    ****)
(****                                                             ****)
(****  Last modification Sep 89                                   ****)
(*************************************************************************)

(* Modification history

   8 Sep 89  - added PROMPT to the list of internal commands
   24 Mar 90 - deleted Find_Environment (duplicated in Unit Support
*)

UNIT Spawn;

INTERFACE

usee Datacom, Dos, Crt, Redirect, Support, ErrorCod, Miscpack;

TYPE
  Internal_Command = (CD, CHDIR, COPI, DEL, DIR, ERASE, EQUIP, LS, MD,
                      MKDIR, PROMT, RD, REN, RENAME, RMDIR );

CONST
  Command_Name : Array [Internal_Command] OF String[6] =
                  ('CD', 'CHDIR', 'COPY', 'DEL', 'DIR',
                   'ERASE', 'EQUIP', 'LS', 'MD', 'MKDIR', 'PROMPT',
                   'RD', 'REN', 'RENAME', 'RMDIR' );

  Com_Port : String[6] = '[COM1]';
```

```
VAR
  Redirection : boolean; { set by the caller in the main program to
                           force all command program output to file
                           for remote display }

FUNCTION Match_Command ( VAR FileSpec : FileString;
                         VAR Command  : Internal_Command ) : boolean;

{ Matches the command in FileSpec against the above list of commands
  processed internally by this program.

  Input:  FileSpec is the command/file name

  Output: FileSpec is adjusted to contain the complete path, if any
          Command is an enumerated type for internal commands
          The function returns true if a command is matched

}

Procedure Run_Local (     ProgramName, Cmdline : string;
                      VAR Response  : string128;
                      VAR Restype   : Response_type;
                      VAR Error_msg : string128;
                      VAR Errtype   : Response_type;
                      VAR Prompt    : string128;
                          Batch     : boolean );

{ Used to spawn a child process, program name in Command,
  parameters in Command_Tail.  Program output, error responses,
  and a follow on command line prompt as it would appear from a
  local command line processor are returned to the calling
  program.

  Input:  ProgramName is the command to be executed with path
          Cmdline is the command tail for ProgramName
          Batch lets Run_Local know a batch file is to be executed

  Output: Response is the output of the program
          Restype is the type of Response (string, file, nothing)
          Error_Msg is the error output of the program
          Errtype is the type of Error_Msg (string, file, nothing)
          Prompt is a simulated command line prompt after program
          execution

}
```

```
PROCEDURE Process_intrinsic_command ( Command : Internal_command;
                                      Command_tail  : String128;
                                      VAR Response  : String128;
                                      VAR Restype   : Response_type;
                                      VAR Error_msg : String128;
                                      VAR Errtype   : Response_type;
                                      VAR Prompt    : String128 );

{ Used to execute a command normally processed internally by
  command.com.  The program name is found in Command,
  parameters in Command_Tail.  Program output, error responses,
  and a follow on command line prompt as it would appear from a
  local command line processor are returned to the calling
  program.

  Input:  Command is the command to be executed with path
          Command_Tail is the command tail for Command

  Output: Response is the output of the program
          Restype is the type of Response (string, file, nothing)
          Error_Msg is the error output of the program
          Errtype is the type of Error_Msg (string, file, nothing)
          Prompt is a simulated command line prompt after program
          execution

}

IMPLEMENTATION

FUNCTION Match_Command ( VAR FileSpec : FileString;
                         VAR Command  : Internal_Command ) : boolean;

{ Matches the command in FileSpec against the above list of commands
  processed internally by this program.  Returns true if a command
  is matched

  Input:  FileSpec is the command/file name

  Output: FileSpec is adjusted to contain the complete path, if any
          Command is an enumerated type for internal commands
          The function returns true if a command is matched

}

VAR
  Found : boolean;
  index : Internal_Command;
```

```
BEGIN
  Found := FALSE;
  FOR index := CD TO RMDIR DO

    IF ( Pos ( Command_Name[ index ], FileSpec ) = 1 ) AND
       ( Length ( Command_Name[ index ] ) = Length ( FileSpec )) THEN
         BEGIN
           Found := TRUE;
           Command := index;
         END;
  Match_Command := Found;
END;

Procedure Run_Local (     ProgramName, Cmdline : string;
                      VAR Response  : string128;
                      VAR Restype   : Response_type;
                      VAR Error_msg : string128;
                      VAR Errtype   : Response_type;
                      VAR Prompt    : string128;
                          Batch : boolean );


{ Used to spawn a child process, program name in Command,
  parameters in Command_Tail.  Program output, error responses,
  and a follow on command line prompt as it would appear from a
  local command line processor are returned to the calling
  program.

  The use of a secondary copy of COMMAND.COM to run batch files is from
  (Mefford, 1988, p. 327).

  Input:  ProgramName is the command to be executed with path
          Cmdline is the command tail for ProgramName
          Batch lets Run_Local know a batch file is to be executed

  Output: Response is the output of the program
          Restype is the type of Response (string, file, nothing)
          Error_Msg is the error output of the program
          Errtype is the type of Error_Msg (string, file, nothing)
          Prompt is a simulated command line prompt after program
          execution

}

begin
  CheckBreak := TRUE;
  IF Batch THEN BEGIN
    Cmdline := ' /c ' + programname + Cmdline;
    { set up temporary command.com }
    ProgramName := Find_Environment ( 'COMSPEC');
  END;
```

```
  GetDir(0, Prompt);
  IF Redirection THEN BEGIN
    Init_Redirect_Unit;
    IF Redirect_All_Output THEN;
  END;
  Exec (Programname, Cmdline);
  IF Redirection THEN BEGIN
    IF Restore_All_Output THEN;
    Restore_CRT_Assignments;
  END;
  RS_Cleanup;
  RS_Restore ( Current_COM );
  Restype := file_type;
  Response := Redirect.Response_file;
  Errtype := strng;
  IF doserror <> 0 THEN BEGIN
     Error_Msg := Error_Code [ DosError ];
   END
   else Error_Msg := '';
  System.ChDir ( Prompt );
  Prompt := Prompt + '>';
END;


CONST
  SPACE : Char = ' ';

PROCEDURE Process_intrinsic_command ( Command : Internal_command;
                                      Command_tail  : String128;
                                      VAR Response  : String128;
                                      VAR Restype   : Response_type;
                                      VAR Error_msg : String128;
                                      VAR Errtype   : Response_type;
                                      VAR Prompt    : String128 );


{ Used to execute a command normally processed internally by
  command.com.  The program name is found in Command,
  parameters in Command_Tail.  Program output, error responses,
  and a follow on command line prompt as it would appear from a
  local command line processor are returned to the calling
  program.

  Input:  Command is the command to be executed with path
          Command_Tail is the command tail for Command

  Output: Response is the output of the program
          Restype is the type of Response (string, file, nothing)
          Error_Msg is the error output of the program
          Errtype is the type of Error_Msg (string, file, nothing)
          Prompt is a simulated command line prompt after program
          execution. }
```

```
CONST Current_Drive : byte = 0;
      Batch_mode    : boolean = TRUE;

VAR  IOR : word;
     Current_Path : PathString;
     List : EquipmentListType;

BEGIN
  CASE Command OF

    CD,
    MD,
    RD,
    CHDIR,
    MkDir,
    Promt,
    RmDir : BEGIN
              {$I-}
              Restype := strng;
              Errtype := strng;

              CASE Command OF

                CD,
                ChDir : System.ChDir ( Command_tail );

                MD,
                MKDIR : System.MkDir ( Command_tail );

                PROMT : GetDir ( Current_Drive, Prompt );

                RD,
                RMDIR : System.RmDir ( Command_tail );

              END;
              IOR := IOResult;
              IF IOR <> 0 THEN
                Error_msg := ^G + Error_Code [ IOR ]
              ELSE Error_msg := '';
              GetDir ( Current_Drive, Prompt );
              Response := '';
              Prompt := Prompt + '>';
              {$I+}
            END;
```

```
    DEL,
    LS,
    DIR,
    REN,
    COPI,
    ERASE,
    RENAME : BEGIN
              IF Command = LS THEN Command := DIR;
              Run_Local ( Command_Name [ Command ] + SPACE,
                          Command_Tail,
                          Response, Restype, Error_msg, Errtype,
                          Prompt, Batch_Mode );
             END;

    EQUIP  : begin
               CheckBreak := TRUE;
               GetDir(0, Prompt);
               IF Redirection THEN BEGIN
                 Init_Redirect_Unit;
                 IF Redirect_All_Output THEN;
               END;

               Support.GetEquip ( List );

               Errtype := strng;
               IF Redirection THEN BEGIN
                 IF Restore_All_Output THEN;
                 Restore_CRT_Assignments;
                 Restype := file_type;
                 Response := Redirect.Response_file;
                 IF doserror <> 0 THEN
                   Error_Msg := Error_Code [ DosError ]
                 else Error_Msg := '';
                 END
               ELSE BEGIN
                 Restype := strng;
                 Response := 'Unable';
                 Error_Msg := '';
               END;
               System.ChDir ( Prompt );
               Prompt := Prompt + '>';
             END;

  END; {CASE}
END;

BEGIN
  Redirection := TRUE; { output is normally redirected to file }
end.
```

{

APPENDIX X

SOURCE LISTING FOR UNIT SUPPORT

```
}
(*************************************************************)
(****                    SUPPORT.PAS                      ****)
(****  This is the unit that contains typed constants for use ****)
(****  by the main program Distrib to display window menus.  ****)
(****  In addition to general purpose routines, the unit also ****)
(****  contains the initialization procedure for the program. ****)
(****                                                       ****)
(****                                                       ****)
(****  References: Edwards, C. C., Advanced Techniques in Turbo ****)
(****              Pascal, pp. 241 - 272, Sybex, Inc., 1987 ****)
(****                                                       ****)
(****              Hall, W.V., "When Turbo Isn't Enough," in ****)
(****              Shammas, N.C., The Turbo Pascal Toolbook, ****)
(****              pp. 225 - 226, M & T Publishing, Inc., 1986. ****)
(****                                                       ****)
(****  Converted to a unit from program Turbocom.com in the ****)
(****  first reference.                                     ****)
(****                                                       ****)
(****                                                       ****)
(****  Last modification Sep 89                             ****)
(*************************************************************)

UNIT Support;

{  Modification History
   4 Aug 89 - Changed introductory maintenance screen
              Deleted conversion messages from TP 4.0 }

INTERFACE

(*********************  Start Edwards Excerpt  *********************)

Uses
  Crt,
  Dos,
  General,
  Datacom,
  Wndow,
  Printer;
(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)
Const Alt_A = 30;
      Alt_B = 48;
      Alt_C = 46;
      Alt_D = 32;
      Alt_E = 18;
      Alt_F = 33;
      Alt_G = 34;
      Alt_H = 35;
      Alt_I = 23;
      Alt_J = 36;
      Alt_K = 37;
      Alt_L = 38;
      Alt_M = 50;
      Alt_N = 49;
      Alt_O = 24;
      Alt_P = 25;
      Alt_Q = 16;
      Alt_R = 19;
      Alt_S = 31;
      Alt_T = 20;
      Alt_U = 22;
      Alt_V = 47;
      Alt_W = 17;
      Alt_X = 45;
      Alt_Y = 21;
      Alt_Z = 44;
      Home  = 71;
      PgUp  = 73;
      PgDn  = 81;

Const NUL = $00;
      SOH = $01;
      STX = $02;
      ETX = $03;
      EOT = $04;
      ENQ = $05;
      ACK = $06;
      BEL = $07;
      BS  = $08;
      HT  = $09;
      LF  = $0A;
      VT  = $0B;
      FF  = $0C;
      CR  = $0D;
      SO  = $0E;
      SI  = $0F;
(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  **********************)
```

```
(*********************  Continue Edwards Excerpt  *********************)
      DLE = $10;
      DC1 = $11;
      DC2 = $12;
      DC3 = $13;
      DC4 = $14;
      NAK = $15;
      SYN = $16;
      ETB = $17;
      CAN = $18;
      EM  = $19;
      SUB = $1A;
      ESC = $1B;
      FS  = $1C;
      GS  = $1D;
      RS  = $1E;
      US  = $1F;

      CEE = $43;

Type Phone_Name = String[30];
     Phone_Params = Record
                    Phone_Number:String[20];
                    Phone_Baud:RS_Baud;
                    Phone_Parity:RS_Parity;
                    Phone_Length:Byte;
                    Phone_Stop:Byte;
                    Phone_Echo:Boolean;
                    End;
     Phone_Record = Record
                    Name:Phone_Name;
                    Phone_Data:Phone_Params;
                    End;
     Phone_Names = Record
                   Length:Integer;
                   Names:Array [1..1] of Phone_Name;
                   End;
     Phone_Data = Array [1..1] of Phone_Params;
     Communications_Type = Record
                           Speed:RS_Baud;
                           Parity:RS_Parity;
                           Length:Byte;
                           Stop:Byte;
                           End;
     String3 = String[3];
     String4 = String[4];

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*********************  Continue Edwards Excerpt  *********************)
```

```
(*********************  Continue Edwards Excerpt  *********************)

TYPE EquipmentListType = RECORD
                           NbrOfPrinters,
                           NbrOfSerial,
                           NbrOfDiskettes,
                           InitialVideo,
                           RAMOnBoard : word;
                           IsGamePort,
                           IsDiskette  : boolean;
                         END;
VAR List : EquipmentListType;

Var Phone_File:File of Phone_Record;
{Moved from Dialing Directory }
    Phone_Menu,
    Old_Phone_Menu:^Phone_Names;
    Phone_Stuff,
    Old_Phone_Stuff:^Phone_Data;
    Phone_Prefix:String [10];
    Echo,Print,Ascii_Upload,Ascii_Download,End_Emulator:Boolean;
    Status_Line:String[80];
    Emulator:String[10];
    Ascii_File : File of Char;
    Ascii_FileName:String[20];
    Current_Path:Long_String;
    Dial_Delay:Integer;
    Last_Dial:Integer;

Type Default_Type = Record   {The default parameters for Distrib}
                    Default_Name:String[30];
                    Default_Com:Byte;
                    Default_Modem:Byte;
                    Default_Phone:String[20];
                    Default_Speed:RS_Baud;
                    Default_Parity:RS_Parity;
                    Default_Length:Byte;
                    Default_Stop:Byte;
                    Default_Echo:Boolean;
                    Default_Textcolor:Byte;
                    Default_Menucolor:Byte;
                    Default_Backcolor:Byte;
                    Default_Prefix:String[10];
                    Default_Delay:Integer;
                    End;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

VAR Current : Default_Type;

Const Defaults:Default_Type =
      (Default_Name:'DISTRIB.CFG';
       Default_Com:1;
       Default_Modem:2;
       Default_Phone:'555-1212';
       Default_Speed:B9600;
       Default_Parity:None;
       Default_Length:8;
       Default_Stop:1;
       Default_Echo:False;
       Default_Textcolor:LightGray;
       Default_Menucolor:Green;
       Default_Backcolor:Black;
       Default_Prefix:'ATDT9,,9,,';
       Default_Delay:30);

       OK_Menu:Integer = 1;
       OK_Msg:String[3] = 'OK ';

       Yes_No_Menu:Integer = 2;
       Yes_No_Msg:Array [1..2] of String[3] = (
             'No ',
             'Yes');

       Dial_Menu:Integer = 5;
       Dial_Msg:Array [1..5] of String[6] = (
            'Dial  ',
            'Repeat',
            'Modify',
            'Delete',
            'Add   ');

       Speed_Menu:Integer = 10;
       Speed_Msg:Array [1..10] of String[4] = (
            ' 110',
            ' 150',
            ' 300',
            ' 600',
            '1200',
            '2400',
            '4800',
            '9600',

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

            '19K2',
            '38K4');

      Parity_Menu:Integer = 3;
      Parity_Msg:Array [1..3] of String[4] = (
            'None',
            'Odd ',
            'Even');

      Stop_Menu:Integer = 2;
      Stop_Msg:Array [1..2] of String[6] = (
            '0 Bits',
            '1 Bit ');

      Length_Menu:Integer = 4;
      Length_Msg:Array [1..4] of String[6] = (
            '5 Bits',
            '6 Bits',
            '7 Bits',
            '8 Bits');

      Param_Menu:Integer = 14;
      Param_Msg:Array [1..14] of String[16] = (
            'Name            ',
            'Phone Number    ',
            'Speed           ',
            'Word Length     ',
            'Parity          ',
            'Stop Bits       ',
            'Local Echo      ',
            'Comm Port       ',
            'Modem Port      ',
            'Dial Prefix     ',
            'Redial Delay    ',
            'Foreground Color',
            'Background Color',
            'Menu Color      ');

      Color_Menu:Integer = 8;
      Color_Msg:Array [1..9] of String[7] = (
            'Black  ',
            'Blue   ',
            'Green  ',
            'Cyan   ',

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

            'Red    ',
            'Magenta',
            'Brown  ',
            'White  ',
            'Nothing');

      Comm_Menu:Integer = 2;
      Comm_Msg:Array [1..2] of String[5] = (
            'COM 1',
            'COM 2');

      Protocol_Menu:Integer = 2;
      Protocol_Msg:Array [1..2] of String[6] = (
            'Ascii ',
            'XModem');

      Communications_Menu:Integer = 21;
      Communications_Msg:Array [1..21] of String[10] = (
            ' 300-E-7-1',
            ' 300-O-7-1',
            ' 300-N-8-1',
            '1200-E-7-1',
            '1200-O-7-1',
            '1200-N-8-1',
            '2400-E-7-1',
            '2400-O-7-1',
            '2400-N-8-1',
            '4800-E-7-1',
            '4800-O-7-1',
            '4800-N-8-1',
            '9600-E-7-1',
            '9600-O-7-1',
            '9600-N-8-1',
            '19K2-E-7-1',
            '19K2-O-7-1',
            '19K2-N-8-1',
            '38K4-E-7-1',
            '38K4-O-7-1',
            '38K4-N-8-1');
      Communications_Stuff:Array [1..21] of Communications_Type = (
            (Speed:B300;Parity:Even;Length:7;Stop:1),
            (Speed:B300;Parity:Odd;Length:7;Stop:1),
            (Speed:B300;Parity:None;Length:8;Stop:1),
            (Speed:B1200;Parity:Even;Length:7;Stop:1),
            (Speed:B1200;Parity:Odd;Length:7;Stop:1),

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

          (Speed:B1200;Parity:None;Length:8;Stop:1),
          (Speed:B2400;Parity:Even;Length:7;Stop:1),
          (Speed:B2400;Parity:Odd;Length:7;Stop:1),
          (Speed:B2400;Parity:None;Length:8;Stop:1),
          (Speed:B4800;Parity:Even;Length:7;Stop:1),
          (Speed:B4800;Parity:Odd;Length:7;Stop:1),
          (Speed:B4800;Parity:None;Length:8;Stop:1),
          (Speed:B9600;Parity:Even;Length:7;Stop:1),
          (Speed:B9600;Parity:Odd;Length:7;Stop:1),
          (Speed:B9600;Parity:None;Length:8;Stop:1),
          (Speed:B19200;Parity:Even;Length:7;Stop:1),
          (Speed:B19200;Parity:Odd;Length:7;Stop:1),
          (Speed:B19200;Parity:None;Length:8;Stop:1),
          (Speed:B38400;Parity:Even;Length:7;Stop:1),
          (Speed:B38400;Parity:Odd;Length:7;Stop:1),
          (Speed:B38400;Parity:None;Length:8;Stop:1));

     Help_Menu:Integer = 17;
     Help_Msg:Array [1..17] of String[26] = (
          'Alt-A  Change drive & path',
          'Alt-B  Send a Break signal',
          'Alt-C  Update Config File ',
          'Alt-D  Dialing Directory  ',
          'Alt-E  Local echo toggle  ',
          'Alt-F  Change DC params   ',
          'Alt-G  Show disk directory',
          'Alt-H  Hang up phone      ',
          'Alt-L  DOS Shell          ',
          'Alt-M  Activate Master    ',
          'Alt-P  Port Operations    ',
          'PgDn,                     ',
          'Alt-R  XMODEM Get a file  ',
          'Alt-S  Activate Server    ',
          'PgUp,                     ',
          'Alt-T  XMODEM Put a file  ',
          'Alt-X  (ESC) Exit emulator');
     Help_Index:Array [1..17] of Byte =(
          Alt_A,
          Alt_B,
          Alt_C,
          Alt_D,
          Alt_E,
          Alt_F,
          Alt_G,

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

            Alt_H,
            Alt_L,
            Alt_M,
            Alt_P,
            PgDn,
            Alt_R,
            Alt_S,
            PgUp,
            Alt_T,
            Alt_X);

Procedure Initialize;

Procedure Modify_Entry(I:Integer);

Procedure Save_File(D:Boolean);

Procedure OK(S:String3);

Function Yes(S:String4):Boolean;

Procedure Build_Status_Line;

Function Check_Keyboard:Integer;

Function Check_Auxport:Char;

Function Find_Environment(What:Long_String):Long_String;

Procedure NoFile(S:Long_String);

Procedure GetEquip ( VAR List : EquipmentListType );

IMPLEMENTATION

Procedure Initialize;

{This procedure initializes the default values and reads the phone file}

Var Phone:Phone_Record;
    I:Integer;
    Configuration : File;
    Numread : word;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

   Begin
     Assign( Configuration, Defaults.Default_Name );
{$I-}
     Reset ( Configuration, Sizeof ( Defaults ) );
     If IOResult > 0 then Current := Defaults
     ELSE Begin
       BlockRead ( Configuration, Current, Sizeof ( Defaults ),Numread);
       Close ( Configuration );
       If IOResult > 0 then Current := Defaults;
     End;
{$I+}
   With Current do
      Begin
      ClrScr;
      If not Mono then
         Begin
         SetColor(Default_Textcolor);
         SetBackground(Default_Backcolor);
         Menuground:=Default_Menucolor;
         End;
      Phone_Prefix:=Default_Prefix;
      Echo:=Default_Echo;
      Dial_Delay:=Default_Delay;
      Print:=False;
      Ascii_Upload:=False;
      Ascii_Download:=False;
      GotoXY(27,1);
      Textcolor(Default_Textcolor+8);
      Writeln('Remote Server Version 1.0');
      GotoXY(31,2);
      Write('Maintenance Screen');
      GotoXY(35,3);
      Write('Dr. Kodres');
      Textcolor(Default_Textcolor);
      Write_Status('                                        Initializing',
                   Default_Textcolor shl 4+Default_Backcolor+$80);
      End;
   Last_Dial:=1;
   Assign(Phone_File,'DISTRIB.PHN');
{$I-}
   Reset(Phone_File);



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

{$I+}
   If IOResult > 0 then
      Begin   {Create new file}
      GetMem(Phone_Menu,SizeOf(Phone_Names));
      GetMem(Phone_Stuff,SizeOf(Phone_Params));
      Phone_Menu^.Length:=1;
      Phone_Menu^.Names[1]:='...To be provided...          ';
      Move(Defaults.Default_Phone,Phone_Stuff^[1],SizeOf(Phone_Params));
      Phone.Name:=Phone_Menu^.Names[1];
      Phone.Phone_Data:=Phone_Stuff^[1];
      Rewrite(Phone_File);
      Write(Phone_File,Phone);
      End
   else
      Begin   {Get phone file}
      I:=FileSize(Phone_File);
      GetMem(Phone_Menu,I*SizeOf(Phone_Name)+2);
      GetMem(Phone_Stuff,I*SizeOf(Phone_Params));
      Phone_Menu^.Length:=I;
      I:=1;
{$R-}
      While not Eof(Phone_File) do
         Begin
         Read(Phone_File,Phone);
         Phone_Menu^.Names[I]:=Phone.Name;
         Phone_Stuff^[I]:=Phone.Phone_Data;
         I:=I+1;
         End;
{$R+}
      End;
   Close(Phone_File);
   With Current do
      Begin

        RS_Initialize(Default_Com,Default_Speed,Default_Parity,
                      Default_Stop,Default_Length);

      End;
   Write_Status(' ',Current.Default_Backcolor shl 4 +
                Current.Default_Textcolor);
   End;  {of Initialize}



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Procedure Save_File(D:Boolean);

{This procedure asks the user if he wants to save a changed
configuration
 If so, it writes the appropriate file

 Input D: True if saving default values
          False if saving phone file
}
Var Configuration : File;
    Phone:Phone_Record;
    J:Integer;

   Begin
   If Open_Window(50,9,67,12,Flag_Borders,'') = 0 then;
   ClrScr;
   If D then
      Write('Save defaults?')
   else
      Write('Save this entry?');
   If Yes('Save') then
     Begin
     ClrScr;
     Write('Saving...');
     If D then
        Begin
        Assign( Configuration, Defaults.Default_Name );
{$I-}
        Rewrite ( Configuration, Sizeof ( Defaults ) );
        If IOResult > 0 then
           NoFile(Defaults.Default_Name)
        else
           Begin
             BlockWrite ( Configuration, Current, 1 );
             Close ( Configuration );
           End;
        End
{$I+}
     else
        Begin
{$R-}
        Assign(Phone_File,'DISTRIB.PHN');
        Rewrite(Phone_File);


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

        For J:=1 to Phone_Menu^.Length do
           Begin
           Phone.Name:=Phone_Menu^.Names[J];
           Phone.Phone_Data:=Phone_Stuff^[J];
           Write(Phone_File,Phone);
           End;
        Close(Phone_File);
{$R+}
        End;
     End;
   If Close_Window then;
   End;  {of Save_File}

Procedure Modify_Entry(I:Integer);

{This procedure modifies an entry in the phone list.

 Input: I - If > 0 then the entry in the phone list to be modified
            If = 0 then the default parameters
}
Var J,K:Integer;
    Status_Window,Menu_Window:Byte;
    S:Long_String;
    B:Boolean;

    Procedure Update_Status;
    Var J:Integer;
       Begin
{$R-}
       If Get_Window(Status_Window) then;
       For J:=1 to Param_Menu do
          Begin
          GotoXY(18,J);
          ClrEol;
          Case J of
            1:  If I = 0 then
                   Write(Current.Default_Name)
                else
                   Write(Phone_Menu^.Names[I]);
            2:  If I = 0 then
                   Write(Current.Default_Phone)
                else
                   Write(Phone_Stuff^[I].Phone_Number);


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

            3:  If I = 0 then
                   Write(Speed_Msg[Ord(Current.Default_Speed)+1])
                else
                   Write(Speed_Msg[Ord(Phone_Stuff^[I].Phone_Baud)+1]);
            4:  If I = 0 then
                   Write(Length_Msg[Current.Default_Length-4])
                else
                   Write(Length_Msg[Phone_Stuff^[I].Phone_Length-4]);
            5:  If I = 0 then

                   Write(Parity_Msg[Min(Ord(Current.Default_Parity)
                         + 1,3)])
                else
                Write(
                  Parity_Msg[Min(Ord(Phone_Stuff^[I].Phone_Parity)
                    +1,3)]);
            6:  If I = 0 then
                   Write(Stop_Msg[Current.Default_Stop+1])
                else
                   Write(Stop_Msg[Phone_Stuff^[I].Phone_Stop+1]);
            7:  If I = 0 then
                   Write(Yes_No_Msg[Ord(Current.Default_Echo)+1])
                else
                   Write(Yes_No_Msg[Ord(Phone_Stuff^[I].Phone_Echo)+1]);
            8:  Write(Comm_Msg[Current.Default_Com]);
            9:  Write(Comm_Msg[Current.Default_Modem]);
            10: Write(Current.Default_Prefix);
            11: Write(Current.Default_Delay);

            12: Write(Color_Msg[Current.Default_Textcolor+1]);
            13: Write(Color_Msg[Current.Default_Backcolor+1]);
            14: Write(Color_Msg[Current.Default_Menucolor+1]);
            End;  {of Case}
         End;
      If Get_Window(Menu_Window) then;
      End;  {of Update_Status}

   Begin
   If I = 0 then
      If Mono then
         Param_Menu:=10
      else
         Param_Menu:=13


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

   else
      Param_Menu:=7;
   If Open_Window(1,2,50,3+Param_Menu,Flag_Borders,
     'Parameters') = 0 then;
   Status_Window:=Active_Window^.ID;
   ClrScr;
   For J:=1 to Param_Menu do
      Begin
      GotoXY(1,J);
      Write(Param_Msg[J],':');
      End;
   If Open_Window(52,2,70,3+Param_Menu,Flag_Borders,
     'Options') = 0 then;
   Menu_Window:=Active_Window^.ID;
   ClrScr;
   Repeat Begin
          Update_Status;
          J:=Process_Window_Menu(Param_Menu);
          Case J of
             0:  ;  {ESC...do nothing}
             1:  Begin  {Change Name}
                 If Open_Window(5,21,75,24,Flag_Borders,
                  'Name') = 0 then;
                 ClrScr;
                 Write('Name: ');
                 Readln(S);
                 If Length(S) > 0 then
                  If I = 0 then
                    Current.Default_Name:=S
                  else
                    Phone_Menu^.Names[I]:=S
                    +'                                  ';
                 If Close_Window then;
                 End;
             2:  Begin  {Phone number}
                 If Open_Window(5,21,75,24,Flag_Borders,
                   'Phone Number') = 0 then;
                 ClrScr;
                 Write('Phone Number: ');
                 Readln(S);
                 If Length(S) > 0 then
                  If I = 0 then
                    Current.Default_Phone:=S


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                else
                   Phone_Stuff^[I].Phone_Number:=S;
                If Close_Window then;
                End;
            3:  Begin  {Speed}
                If Open_Window(69,5,75,14,Flag_Borders,'Baud') = 0
                  then;
                ClrScr;
                K:=Process_Window_Menu(Speed_Menu);
                If K > 0 then
                 If I = 0 then
                   Current.Default_Speed:=RS_Baud(K-1)
                 else
                   Phone_Stuff^[I].Phone_Baud:=RS_Baud(K-1);
                If Close_Window then;
                End;
            4:  Begin  {Word Length}
                If Open_Window(69,6,77,11,Flag_Borders,'Bits') = 0
                  then;
                ClrScr;
                K:=Process_Window_Menu(Length_Menu);
                If K > 0 then
                 If I = 0 then
                   Current.Default_Length:=K+4
                 else
                   Phone_Stuff^[I].Phone_Length:=K+4;
                 If CLose_Window then;
                 End;
            5:   Begin  {Parity}
                 If Open_Window(69,7,75,11,Flag_Borders,
                   'Type') = 0 then;
                 ClrScr;
                 K:=Process_Window_Menu(Parity_Menu);
                 If K < 3 then K:=K-1;
                 If K >= 0 then
                   If I = 0 then
                      Current.Default_Parity:=RS_Parity(K)
                   else
                      Phone_Stuff^[I].Phone_Parity:=RS_Parity(K);
                 If Close_Window then;
                 End;
            6:   Begin  {Stop bits}
                 If Open_Window(69,8,77,11,Flag_Borders,
                   'Bits') = 0 then;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                ClrScr;
                K:=Process_Window_Menu(Stop_Menu);
                If K > 0 then
                  If I = 0 then
                    Current.Default_Stop:=K-1
                  else
                    Phone_Stuff^[I].Phone_Stop:=K-1;
                If Close_Window then;
                End;
            7:  Begin  {Local echo}
                B:=Yes('Echo');
                If I = 0 then
                   Current.Default_Echo:=B
                else
                   Phone_Stuff^[I].Phone_Echo:=B;
                End;
            8:  Begin  {Comm port}
                If Open_Window(69,10,76,13,Flag_Borders,
                  'Port') = 0 then;
                ClrScr;
                K:=Process_Window_Menu(Comm_Menu);
                If K > 0 then
                   Current.Default_Com:=K;
                If Close_Window then;
                End;
            9:  Begin  {Comm port}
                If Open_Window(69,10,76,13,Flag_Borders,
                  'Port') = 0 then;
                ClrScr;
                K:=Process_Window_Menu(Comm_Menu);
                If K > 0 then
                   Current.Default_Com:=K;
                If Close_Window then;
                End;
           10:  Begin  {Dial Prefix}
                If Open_Window(5,21,75,24,Flag_Borders,
                  'Prefix') = 0 then;
                ClrScr;
                Write('Prefix: ');
                Readln(S);
                If Length(S) > 0 then
                   Current.Default_Prefix:=S;
                If Close_Window then;
                End;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

              11: Begin  {Default redial delay}
                  If Open_Window(5,21,75,24,Flag_Borders,
                    'Redial delay') = 0 then;
                  ClrScr;
                  Write('Redial delay (in seconds): ');
                  Readln(Current.Default_Delay);
                  If Close_Window then;
                  End;
              12,         {Foreground color}
              13,         {Background color}
              14: Begin  {Menu color}
                  If Open_Window(69,2+J,78,11+J,Flag_Borders,
                    'Colors') = 0 then;
                  ClrScr;
                  K:=Process_Window_Menu(Color_Menu);
                  If K > 0 then
                     Case J of
                        12:  Current.Default_Textcolor:=K-1;
                        13:  Current.Default_Backcolor:=K-1;
                        14:  Current.Default_Menucolor:=K-1;
                        End;  {of Case}
                  If Close_Window then;
                  End;
              End;  {of Case}
          End
   Until J = 0;
   If Close_Window then;
   Save_File(I = 0);
   If Close_Window then;
{$R+}
   End;  {of Modify_Entry}

Procedure OK(S:String3);
{This procedure displays a window on the screen and waits for an
 acknoledgement from the user

 Input: S - The title to use for the window
}
   Begin
   If Open_Window(60,5,64,7,Flag_Borders,S) = 0 then;
   If Process_Window_Menu(OK_Menu) = 0 then;
   If Close_Window then;
   End;  {of OK}


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Function Yes(S:String4):Boolean;
{This procedure prompts the user for a yes or no response

 Input: S - The title to use for the window

 Output: True if YES was selected
}
   Begin
   If Open_Window(69,9,74,12,Flag_Borders,S) = 0 then;
   Yes:=Process_Window_Menu(Yes_No_Menu) = 2;
   If Close_Window then;
   End;  {of Yes}

Procedure Build_Status_Line;
{This procedure updates and displays the status line}

VAR Comport : string[1];

   Begin
   Str ( Current_COM, Comport );
   Status_Line:='                                                '+ {40 spaces}
                '                                                '; {40 spaces}
   Insert('Com Port: ',Status_Line,1);
   Insert(Comport,Status_Line,11);
   WITH Datacom.Comport [ Current_COM ] DO
      BEGIN
        Insert ( Speed_Msg[ORD( Speed ) + 1 ], Status_Line, 13);
        Insert ( 'Baud  ', Status_Line, 18 );
        Insert ( Length_Msg[ Length-4], Status_Line, 23 );
        Insert ( Parity_Msg[Min(ORD( Parity )+1, 3)], Status_Line, 30);
        Insert ( Stop_Msg[ Stop + 1], Status_Line, 35 );
      END;
   If Echo then
      Insert('Echo',Status_Line,47);
   If Print then
      Insert('Print',Status_Line,52);
   Insert('Home for Help',Status_Line,68);
   Write_Status(Status_Line,Foreground shl 4 + Background);
   End;  {of Build_Status_Line}

Function Check_Keyboard:Integer;

{This function checks for keyboard input


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Output: 0 if no key pressed
        If normal key then high byte is 0 and low byte is value of key
        If special key then low byte is 0 and high byte is value of key
}
Var Ch:Char;
   Begin
   If Ascii_Upload then
      Begin
      If Eof(Ascii_File) then
         Begin
         Close(Ascii_File);
         Ascii_Upload:=False;
         Build_Status_Line;
         End
      else
         Begin
         Read(Ascii_File,Ch);
         If Ch = Char(LF) then
            Ch:=Char(NUL);
         Check_Keyboard:=Byte(Ch);
         End
      End
   else if Keypressed then
      Begin
      Ch := ReadKey;
      If (Ch = #0) then
         Begin
         Ch := ReadKey;
         Check_Keyboard:=Byte(Ch) shl 8;
         End
      else
         Check_Keyboard:=Byte(Ch);
      End
   else
      Check_Keyboard:=0;
   End;  {of Check_Keyboard}

Function Check_Auxport:Char;
{This function checks for input from the data communications port
 If the appropriate global booleans are set, it will send the output
 to the printer or to a disk file

 Output: NUL if no character otherwise character received
}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Var Ch:Char;
   Begin
   If RS232_Avail then
      Begin
      Ch := RS232_In;
      If Ch <> Char(NUL) then
         Begin
         If Print then
            Write(LST,Ch);
         If Ascii_Download then
            Write(Ascii_File,Ch);
         End;
      Check_Auxport:=Ch;
      End
   else
      Check_Auxport:=Char(NUL);
   End;  {of Check_Auxport}

Function Find_Environment(What:Long_String):Long_String;

{This function searches the environment for a particular specifier of
 the form:  ID=Text

 Input:  What - the ID to look for

 Output: The Text of the environment string or empty if not found
}
   Type Environment = Array [1..32767] of Char;
   Var Environ:^Environment;
       Environ_Segment : word;
       S:Long_String;
       I:Integer;
   Begin
   Environ_Segment := MemW[PrefixSeg:$002C];
   Find_Environment:='';   {Assume not found}
   What:=What+'=';
   Environ:=PTR(Environ_Segment,0);
   I:=1;
   While Environ^[I] <> ^@ Do
      Begin
      S:='';



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      Repeat Begin
             S:=S+Environ^[I];
             I:=I+1;
             End
      Until Environ^[I] = ^@;
      If (Length(S) >= Length(What)) and
         (Copy(S,1,Length(What)) = What) Then
         Find_Environment:=Copy(S,Length(What)+1,Length(S)-Length(What))
      else
         I:=I+1;
      End;
   End; {of Find_Environment}

Procedure NoFile(S:Long_String);

{This procedure opens a window and notifies the user that a file was
 not found}
  Begin
   If Open_Window(42,2,80,5,Flag_Borders,'No file') = 0 then;
   ClrScr;
   Write('Cannot find file ',S);
   OK('');
   If Close_Window then;
   End;  {of NoFile}


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*************************  End Edwards Excerpt  *************************)

(************************  Start Hall Excerpt  ************************)

Procedure GetEquip ( VAR List : EquipmentListType );

CONST SYS_INT : byte = $11;

VAR Regs : Dos.Registers;

BEGIN
  With List DO BEGIN
    With Regs DO BEGIN


(*  The library GetEquip appears in The Turbo Pascal Toolbook by Namir
    C. Shammas (ed.) and has been reprinted with the permission of the
    publisher M & T Books 1-800-533-4372.  Minor modifications by Nelson
    Ard.
*************************  Continue Hall Excerpt  *********************)
```

```
(************************  Continue Hall Excerpt  **********************)
      INTR ( SYS_INT, Regs);
      NbrOfPrinters := AH SHR 6;
      IsGamePort := (AH AND $10) > 1;
      NbrOfSerial := (AH AND $0E) SHR 1;
      IsDiskette := (AL AND $01) = 1;
      IF IsDiskette THEN
        NbrOfDiskettes := (AL SHR 6) + 1
      ELSE
        NbrOfDiskettes := (AL SHR 6) + 0;

      InitialVideo := (AL AND $30)SHR 4;
      CASE InitialVideo OF
        1 : InitialVideo := 0;
        2 : InitialVideo := 2;
        3 : InitialVideo := 7;
      END;
      RAMOnBoard := ((AL AND $0C) + 1) * 16;
    END; { Regs }
    Writeln;
    Writeln ('No. of Printers  = ', NbrOfPrinters );
    Writeln ('No. of Serial    = ', NbrOfSerial );
    Writeln ('No. of Diskettes = ', NbrOfDiskettes );
    Writeln ('InitialVideo = ', InitialVideo );

    Writeln ('RAMOnBoard       = ', RAMOnBoard );
    Writeln ('IsGamePort       = ', IsGamePort );
  END;
END;

(*  The library GetEquip appears in The Turbo Pascal Toolbook by Namir
    C. Shammas (ed.) and has been reprinted with the permission of the
    publisher M & T Books 1-800-533-4372.  Minor modifications by Nelson
    Ard.
*****************************  End Hall Excerpt  ************************)

BEGIN
END.
```

{

APPENDIX Y

SOURCE LISTING FOR UNIT WNDOW

```
}
(*************************************************************************)
(****                        WNDOW.PAS                                ****)
(****  This is a library of general purpose routines to           ****)
(****  display windows and control menu bars for selectors on     ****)
(****  the IBM PC screen.                                         ****)
(****                                                             ****)
(****  Reference:  Edwards, C. C., Advanced Techniques in Turbo ****)
(****              Pascal, pp. 73-97, Sybex, Inc., 1987           ****)
(****                                                             ****)
(****  Modified slightly to make a Turbo Pascal 4.0 Unit          ****)
(****                                                             ****)
(****  Last Modification:  Sep 89                                 ****)
(*************************************************************************)

UNIT Wndow;

INTERFACE

(*********************  Start Edwards Excerpt  *********************)

USES General, Crt, Dos;

{$V-}

Type Window_Link = ^Window_Control_Block;
     Screen_Line = Array [1..80] of WORD;           {! changed per
                                                       upgrade }
     Screen_Array = Array[1..25] of Screen_Line;
     Screen_Block = Array[1..2000] of Integer;
     Window_Title = String[50];
     Window_Control_Block = Record
                            X1,Y1,X2,Y2:Byte; {Window boundaries}
                            X,Y:Byte;         {Cursor location}
                            ID:Byte;
                            Menu_Index:Integer;
                            Menu_TopY:Integer;  {The top item in a menu}
                            Flag:Byte;
                            Foreground,Menuground:Byte;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                              Title:Window_Title;
                              Back_Link:Window_Link;
                              Screen_Contents:Screen_Block;
                              End;  {of Record Window_Control_Block}

     Border_Type = (Single,Double);
     Long_String = STRING[255];

Const Foreground:Byte = LightGray;  {Color within the windows}
      Menuground:Byte = LightGray;  {Color of the menu borders}
      Background:Byte = Black;      {Background color}

{These are the bit values of the field "Flag" in Window_Control_Block}
Const Flag_Borders =    $01;          {Borders on the window}
      Flag_Goto    =    $02;          {Goto to this window is allowed}
      Flag_Relocate=    $04;          {Window may be relocated}
      Flag_Close   =    $08;          {Window may be closed from main
                                        menu}

Var W,
    Active_Window:Window_Link;
    Window_Count:Integer;
    Window_Fixed_Part:Integer;
    Mono:Boolean;
{Forced to assign these variables on the same line - type mismatch }
    Screen,
    Screen_New,Screen_Temp:^Screen_Array;

Procedure SetColor(Color:Byte);
{This procedure sets the forground color}

Procedure SetBackground(Color:Byte);
{This procedure sets the background color}

Procedure Get_Dummy_Screen;
{This procedure changes Screen to point to a dummy screen area on
 the heap}

Procedure Get_Real_Screen;
{This procedure undoes the work of Get_Dummy_Screen}



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Procedure Build_Borders(Lines:Border_Type);

{Purpose:
     This procedure builds a border around a window.

Input:
     Lines:Single = Single line border
           Double = Double line border

Output:
     None }

Function Open_Window(X1,Y1,X2,Y2:Byte;Flag:Byte;
                     Name:Window_Title):Byte;
{Purpose:
     This function opens a window on the screen and places a border
     around it.

 Input:
     X1,X2,Y1,Y2 are the coordinates of the window to be opened.
     Flag is a bit mask of functions allowed in this window
     Name is the title of the window

 Output:
     Open_Window returns a byte as follows:
       0 = Window opened OK
       1 = Invalid window coordinates
       2 = Not enough memory
}

Function Close_Window:Boolean;
{This function closes the currently active window.

Output:
     Returns a True if there is no currently active window.
}

Function Save_Window:Window_Link;
{This procedure saves off the current window & closes it

Output:
     Pointer to the saved window
}


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Function Restore_Window(W:Window_Link):Boolean;
{This procedure re-creates a saved window on the screen}

Function Get_Window(Which:Integer):Boolean;
{This procedure brings window "Which" to the top of the screen}

Function Move_Window(X,Y:Integer):Boolean;
{This procedure moves the current window by "X,Y" locations}

Procedure Write_Status(S:Long_String;Attrib:Integer);
{This procedure writes to line 25 of the display

Input: S = String to be written
       Attrib = Video attribute byte to use
}

Function Process_Window_Menu(Var Menu):Byte;
{This procedure will display and process a menu in the currently
 active window.
 The menu may be longer or shorter than the actual window.

 Input:  Menu - A pointer to a record with the following format:
                Bytes 0-1: An integer giving the number of string
                           variables
                Bytes 2-n: A series of String variables.

 Outout: The function returns the index (1 relative) of the item
         selected.  A zero is returned if the ESC key is pressed
}

Procedure Init_Window_Info;
{This procedure initializes all the of data used by the
 windowing routines}

IMPLEMENTATION

Procedure SetColor(Color:Byte);
{This procedure sets the forground color

Input: Color = Color to set forground to }
     Begin
     Foreground:=Color;
     Textcolor(Color);
     End;  {of SetColor}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Procedure SetBackground(Color:Byte);
{This procedure sets the background color

Input: Color = Color to set background to
}
     Begin
     Background:=Color;
     Textbackground(Color);
     End;  {of SetBackground}

Procedure Get_Dummy_Screen;
{This procedure changes Screen to point to a dummy screen area on
 the heap}
     Begin
     If Screen_New <> Nil then
          Begin
          Screen_New^:=Screen^;
          Screen:=Screen_New;
          End;
     End;  {of Get_Dummy_Screen}

Procedure Get_Real_Screen;
{This procedure undoes the work of Get_Dummy_Screen}
     Begin
     If Screen_New <> Nil then
          Begin
          Screen_Temp^:=Screen_New^;
          Screen:=Screen_Temp;
          End;
     End;  {of Get_Real_Screen}

Procedure Build_Borders(Lines:Border_Type);

{Purpose:
     This procedure builds a border around a window.

Input:
     Lines:Single = Single line border
           Double = Double line border

Output:
     None }

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Const Upper_Left:Array [0..1] of Char = (#218,^J);
      Upper_Right:Array [0..1] of Char = (#191,#187);
      Lower_Left:Array [0..1] of Char = (#192,#200);
      Lower_Right:Array [0..1] of Char = (#217,#188);
      Vertical:Array [0..1] of Char = (#179,#186);
      Horizontal:Array [0..1] of Char = (#196,#205);

Var Index:Byte Absolute Lines;
    XX,YY,I:Byte;
    MG,H,V:Integer;
     Begin
     I:=1;
     With Active_Window^ do
        Begin
        If (Flag and Flag_Relocate) = Flag_Relocate then
           Upper_Left[1]:=^J
        else
           Upper_Left[1]:=#201;
        MG:=Menuground shl 8;
        H:=MG+Byte(Horizontal[Index]);
        V:=MG+Byte(Vertical[Index]);
        Screen^[Y1,X1]:=(MG)+Byte(Upper_Left[Index]);
        Screen^[Y1,X2]:=(MG)+Byte(Upper_Right[Index]);
        Screen^[Y2,X1]:=(MG)+Byte(Lower_Left[Index]);
        Screen^[Y2,X2]:=(MG)+Byte(Lower_Right[Index]);
        XX:=X1+1;
        While XX < X2 do
           Begin
           If I <= Length(Title) then
              Screen^[Y1,XX]:=(Foreground shl 8)+Byte(Title[I])
                              + Index shl 11
           else
              Begin
              FillWord(Screen^[Y1,XX],X2-XX,H);
              XX:=X2;
              End;
           XX:=XX+1;
           I:=I+1;
           End;
        FillWord(Screen^[Y2,X1+1],X2-X1-1,H);



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

        For YY:=Y1+1 to Y2-1 do
           Begin
           Screen^[YY,X1]:=V;
           Screen^[YY,X2]:=V;
           End;
        End;  {of With}
     End;  {of Build_Borders}

Function Open_Window(X1,Y1,X2,Y2:Byte;Flag:Byte;
                     Name:Window_Title):Byte;
{Purpose:
     This function opens a window on the screen and places a border
     around it.

 Input:
     X1,X2,Y1,Y2 are the coordinates of the window to be opened.
     Flag is a bit mask of functions allowed in this window
     Name is the title of the window

 Output:
     Open_Window returns a byte as follows:
       0 = Window opened OK
       1 = Invalid window coordinates
       2 = Not enough memory
}
Var Block:Window_Link;
    Line_Length,Window_Size,I:Integer;
    Y,Borders:Byte;

     Begin
     If Active_Window <> Nil then
       If Active_Window^.Flag and Flag_Borders = Flag_Borders then
        Build_Borders(Single);
     Line_Length:=(X2-X1+1);
     Borders:=Byte(Flag and Flag_Borders = Flag_Borders);
     Window_Size:=Line_Length*(Y2-Y1+1)*2+Window_Fixed_Part;
     If (X1 < 1) or (X2 > 80) or (Y1 < 1) or (Y2 > 25) or
        (X2-X1 < 2) or (Y2-Y1 < 2) then
       Open_Window:=1
     else if (MemAvail < Window_Size+1) and (MemAvail >= 0) then
       Open_Window:=2
     else Begin
          GetMem(Block,Window_Size);
          Block^.X1:=X1;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

          Block^.X2:=X2;
          Block^.Y1:=Y1;
          Block^.Y2:=Y2;
          Block^.X:=WhereX;
          Block^.Y:=WhereY;
          Block^.Title:=Name;
          Block^.Flag:=Flag;
          Block^.Menu_Index:=0;
          Block^.Menu_TopY:=0;
          Block^.Foreground:=Foreground+(Background shl 4);
          Block^.Menuground:=Menuground+(Background shl 4);
          Block^.Back_Link:=Active_Window;
          Active_Window:=Block;
          I:=1;
          For Y:=Y1 to Y2 Do
               Begin
               Move(Screen^[Y,X1],Block^.Screen_Contents[I],
                    Line_Length*2);
               I:=I+Line_Length;
               End;
        Window
(X1+Borders,Y1+Borders,X2-Borders,Max((Y2-Borders),(Y1+Borders+1)));
          If Borders = 1 then
              Build_Borders(Double);
          GotoXY(1,1);
          Window_Count:=Window_Count+1;
          Block^.ID:=Window_Count;
          Open_Window:=0;
          End;
     End;  {of Open_Window}

Function Close_Window:Boolean;
{This function closes the currently active window.

Output:
     Returns a True if there is no currently active window.
}

Var Block:Window_Link;
    Line_Length,Window_Size,I:Integer;
    Y,Borders:Byte;
     Begin
     If Active_Window = Nil then
        Close_Window:=True

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

     else
        Begin
        Block:=Active_Window;
        Line_Length:=(Block^.X2-Block^.X1+1);
        Window_Size:=Line_Length*(Block^.Y2-Block^.Y1+1)*2
                     + Window_Fixed_Part;
        I:=1;
        For Y:=Block^.Y1 to Block^.Y2 Do
          Begin
          Move(Block^.Screen_Contents[I],Screen^[Y,Block^.X1],
               Line_Length*2);
          I:=I+Line_Length;
          End;
        Active_Window:=Block^.Back_Link;
        If Active_Window = Nil then
           Window(1,1,80,25)
        else with Active_Window^ do
           Begin
           Borders:=Byte(Flag and Flag_Borders = Flag_Borders);

Window(X1+Borders,Y1+Borders,X2-Borders,Max((Y2-Borders),
      (Y1+Borders+1)));
           If Borders = 1 then
                Build_Borders(Double);
           SetColor(Foreground and 7);
           SetBackground(Foreground shr 4);
           End;
        GotoXY(Block^.X,Block^.Y);
        FreeMem(Block,Window_Size);
        Window_Count:=Window_Count-1;
        Close_Window:=False;
        End;
     End;  {of Close_Window}

Function Save_Window:Window_Link;
{This procedure saves off the current window & closes it

Output:
     Pointer to the saved window
}
Var W:Window_Link;
     Begin
     W:=Active_Window;


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      If Open_Window(W^.X1,W^.Y1,W^.X2,W^.Y2,W^.Flag,W^.Title) > 0 then
        Save_Window:=Nil
     else
        Begin
        Active_Window^.ID:=W^.ID;
        Active_Window^.Menu_Index:=W^.Menu_Index;
        Active_Window^.Menu_TopY:=W^.Menu_TopY;
        W:=Active_Window;
        Active_Window:=W^.Back_Link;
        If Close_Window then;
        Save_Window:=W;
        End;
     End;  {of Save_Window}

Function Restore_Window(W:Window_Link):Boolean;
{This procedure re-creates a saved window on the screen}
     Begin
     SetColor(W^.Foreground and 7);
     SetBackground(W^.Foreground shr 4);
     If Open_Window(W^.X1,W^.Y1,W^.X2,W^.Y2,W^.Flag,W^.Title) > 0 then
        Restore_Window:=True
     else
        Begin
        Active_Window^.ID:=W^.ID;
        Active_Window^.Menu_Index:=W^.Menu_Index;
        Active_Window^.Menu_TopY:=W^.Menu_TopY;
        W^.Back_Link:=Active_Window;
        Active_Window:=W;
        Restore_Window:=Close_Window;
        End;
     End;  {of Restore_Window}

Function Get_Window(Which:Integer):Boolean;
{This procedure brings window "Which" to the top of the screen}
Var WindowP:Window_Link;
   Function Move_Windows:Boolean;
   Var W:Window_Link;
        Begin
        W:=Save_Window;
        If W = Nil then
           Move_Windows:=True
        Else
           If W^.ID <> Which then


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

              Begin
              If Move_Windows then
                 Move_Windows:=True
              else
                 Move_Windows:=Restore_Window(W);
              End
           else
              Begin
              WindowP:=W;
              Move_Windows:=False;
              End;
        End;  {of Move_Windows}
   Begin  {Outer block of Get_Window}
   Get_Window:=False;
   WindowP:=Active_Window;
   While (WindowP <> Nil) and (WindowP^.ID <> Which) do
      WindowP:=WindowP^.Back_Link;
   If WindowP = Nil then
      Get_Window:=True
   else if Active_Window^.ID <> Which then
        Begin
        Get_Dummy_Screen;
        If Move_Windows then
           Get_Window:=True
        else
           Get_Window:=Restore_Window(WindowP);
        Get_Real_Screen;
        End;
   End;  {of Get_Window}

Function Move_Window(X,Y:Integer):Boolean;
{This procedure moves the current window by "X,Y" locations}
Var W:Window_Link;
    XC,YC,Line_Length,YI,Borders:Byte;
    I:Integer;
     Begin
     W:=Active_Window;
     If W = Nil then
        Move_Window:=True
     else if (W^.X1+X < 1) or (W^.Y1+Y < 1) or (W^.X2+X > 80)
                           or (W^.Y2+Y > 24)
        then Move_Window:=True
     else Begin
          XC:=WhereX;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

          YC:=WhereY;
          Line_Length:=W^.X2-W^.X1+1;
          I:=1;
          For YI:=W^.Y1 to W^.Y2 do
             Begin
             Exchange(W^.Screen_Contents[I],Screen^[YI,W^.X1],
                      Line_Length*2);
             I:=I+Line_Length;
             End;
          W^.X1:=W^.X1+X;
          W^.Y1:=W^.Y1+Y;
          W^.X2:=W^.X2+X;
          W^.Y2:=W^.Y2+Y;
          I:=1;
          For YI:=W^.Y1 to W^.Y2 do
             Begin
             Exchange(W^.Screen_Contents[I],Screen^[YI,W^.X1],
                      Line_Length*2);
             I:=I+Line_Length;
             End;
          Borders:=Byte(W^.Flag and Flag_Borders = Flag_Borders);
          Window(W^.X1+Borders,W^.Y1+Borders,W^.X2-Borders,
                 Max((W^.Y2-Borders),(W^.Y1+Borders+1)));
          GotoXY(XC,YC);
          End;
     End;  {of Move_Window}

Procedure Write_Status(S:Long_String;Attrib:Integer);
{This procedure writes to line 25 of the display

Input: S = String to be written
       Attrib = Video attribute byte to use
}
Var X:Byte;
     Begin
     Attrib:=Attrib shl 8;
     For X:=1 to 80 do
        If X > Length(S) then
          Screen^[25,X]:=Attrib+$20
        else
          Screen^[25,X]:=Attrib+Byte(S[X]);
     End;  {of Write_Status}


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Function Keyin (Checkit:Boolean):Integer;
{This procedure reads in a key from the keyboard.

Input: Checkit = True if we should call Special_Processing to check it
               = False if we should not call Special_Processing

Output: The value of the key read
        Function keys are returned with a 0 in the low byte and the
        extended scan code in the high byte
}
Var C:Char;
    Key:Integer;
    Done:Boolean;
    Begin
    Done:=True;
    Repeat
       Begin
       Repeat until KeyPressed;
       C := ReadKey;
       If (C = #0) then
          Begin
          C := ReadKey;
          Key:=Byte(C) shl 8;
          End
       else
          Key:=Byte(C);
       If Checkit then
          Done:=TRUE;
       End
    until Done;
    Keyin:=Key;
    End;  {of Keyin}

Function Process_Window_Menu(Var Menu):Byte;
{This procedure will display and process a menu in the currently
 active window.
 The menu may be longer or shorter than the actual window.

 Input:  Menu - A pointer to a record with the following format:
                Bytes 0-1: An integer giving the number of string
                           variables
                Bytes 2-n: A series of String variables.



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

 Output: The function returns the index (1 relative) of the item
         selected.  A zero is returned if the ESC key is pressed
}
Var Menu_Count:^Integer;
    Menu_Item:^Long_String;
    Menu_Offset:Integer Absolute Menu_Item;
    Window_Size,I,J,Key:Integer;
    Done:Boolean;
        Procedure GoUp;
        {This procedure moves up to the prior item in the menu}
        Begin
        Menu_Offset:=Menu_Offset-Length(Menu_Item^)-1;
        I:=I-1;
        If I < Active_Window^.Menu_TopY then
           Begin
           GotoXY(1,1);
           InsLine;
           Write(Menu_Item^);
           Active_Window^.Menu_TopY:=I;
           End;
        End;  {of GoUp}
        Procedure GoDown;
        {This procedure moves down to the next item in the menu}
        Begin
        Menu_Offset:=Menu_Offset+Length(Menu_Item^)+1;
        I:=I+1;
        If I = Active_Window^.Menu_TopY+Window_Size then
           Begin
           GotoXY(1,1);
           DelLine;
           GotoXY(1,Window_Size);
           Write(Menu_Item^);
           Active_Window^.Menu_TopY:=Active_Window^.Menu_TopY+1;
           End;
        End;  {of GoUp}
        Procedure GoHome;
        {This procedure positions the cursor in the home position}
        Begin
        While I > 1 do
           GoUp;
        End;  {of GoHome}
        Procedure GoEnd;
        {This procedure positions the cursor in the end position}
        Begin

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

        While I < Menu_Count^ do
           GoDown;
        End;  {of GoEnd}
     Procedure Set_Highlights;
        Begin
        With Active_Window^ do
           Begin
           If I = Menu_Index then
              Begin
              Textcolor(Foreground shr 4);
              Textbackground(Foreground and 7);
              End
           else if I = Abs(Menu_Index) then
              Begin
              Textcolor(Blue);
              TextBackground(Black);
              End
           else
              Begin
              Textcolor(Foreground and 7);
              TextBackground(Foreground shr 4);
              End;
           End;
        End;  {of Set_Highlights}
     Begin
     Menu_Count:=Addr(Menu);
     Menu_Item:=Ptr(Seg(Menu),Ofs(Menu)+2);
     Window_Size:=Active_Window^.Y2-Active_Window^.Y1-1;
     If Active_Window^.Menu_Index <= 0 then
        Begin
        ClrScr;
        Active_Window^.Menu_TopY:=1;
        For I:=1 to Min(Menu_Count^,Window_Size) do
           Begin
           GotoXY(1,I);
           Set_Highlights;
           Write(Menu_Item^);
           Menu_Offset:=Menu_Offset+Length(Menu_Item^)+1;
           End;
        If Window_Size = 1 then
           Build_Borders(Double);
        End;
     Menu_Item:=Ptr(Seg(Menu),Ofs(Menu)+2);


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

     For I:=1 to Active_Window^.Menu_Index-1 do
        Menu_Offset:=Menu_Offset+Length(Menu_Item^)+1;
     I:=Max(Active_Window^.Menu_Index,1);
     Active_Window^.Menu_Index:=Min(Active_Window^.Menu_Index,0);
     Done:=False;
     Write_Status('Choose item using the arrow keys '^X' & '^Y' '
                  + #179' Press ESC to abort '+
                  #179' Press '^Q#217' when done',Foreground shl 4);
     Repeat Begin
            TextColor(Active_Window^.Foreground shr 4);
            TextBackground(Active_Window^.Foreground and 7);
            GotoXY(1,I-Active_Window^.Menu_TopY+1);
            Write(Menu_Item^);
            Set_Highlights;
            GotoXY(1,I-Active_Window^.Menu_TopY+1);
            Cursor_Size(Cursor_Invisible,Mono);
            Key:=Keyin(True);
            Write(Menu_Item^);
            Case Lo(Key) of
                0:   Case Hi(Key) of
                        72:  If I > 1 then
                                GoUp
                             else
                                GoEnd;
                        80:  If I < Menu_Count^ then
                                GoDown
                             else
                                GoHome;
                        73:  For J:=1 to Window_Size do
                               If I > 1 then
                                  GoUp;
                        81:  For J:=1 to Window_Size do
                               If I < Menu_Count^ then
                                  GoDown;
                        71:  GoHome;
                        79:  GoEnd;
                        Else Beep(100);
                        End;  {of case}
                13:  Begin
                     Process_Window_Menu:=I;
                     Done:=True;
                     End;


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                27:  Begin
                     Process_Window_Menu:=0;
                     Done:=True;
                     End;
                Else Beep(100);
                End;  {of case}
             End
     Until Done;
     With Active_Window^ do
        Begin
        Menu_Index:=I;
        TextColor(Foreground and 7);
        TextBackground(Foreground shr 4);
        End;
     Write_Status('',Foreground);
     Cursor_Size(Cursor_Small,Mono);
     End;  {of Process_Window_Menu}

Procedure Init_Window_Info;
{This procedure initializes all the of data used by the
 windowing routines}
Var Regs:Registers;
     Begin
     Intr($11,Regs);
     Mono:=(Lo(Regs.AX) and $30) = $30;
     If Mono then
        Screen:=Ptr($B000,0)
     else
        Screen:=Ptr($B800,0);
     Active_Window:=Nil;
     Screen_Temp:=Screen;
     Window_Fixed_Part:=Sizeof(Window_Control_Block)
       - Sizeof(Screen_Block);
     If (MemAvail < 0) or (MemAvail > Sizeof(Screen_Array)+100) then
       { Changed per upgrade to accomodate TP 4.0 MemAvail }
           New(Screen_New)
     else
           Screen_New:=Nil;
     Window_Count:=0;
     End;  {of Init_Window_Info}
BEGIN
END.


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  End Edwards Excerpt  ********************)
```

{

APPENDIX Z

SOURCE LISTING FOR UNIT XMODM

```
}
(*************************************************************)
(****                      XMODM.PAS                      ****)
(****  This is the unit that abstracts all packet and file ****)
(****  transfers for the Xmodem protocol.  The interface is ****)
(****  derived from the Turbocom.com program in the first   ****)
(****  reference, however, the implementation has been rebuilt ****)
(****  for command and data transfer from the second source. ****)
(****                                                       ****)
(****  References: Edwards, C. C., Advanced Techniques in Turbo ****)
(****              Pascal, pp. 220-275, Sybex, Inc., 1987   ****)
(****                                                       ****)
(****              Krantz, D., "Christensen Protocols in C," ****)
(****              Dr. Dobb's Journal, v. 10, no. 6, pp. 66-89, ****)
(****              June 1985.                               ****)
(****                                                       ****)
(****  Modified by Nelson Ard                               ****)
(****                                                       ****)
(****  Last Modification:  Sep 89                           ****)
(*************************************************************)

UNIT Xmodm;

INTERFACE

USES Miscpack, General, Wndow, Datacom, Support, Crt;

{  13 Jun 89 - changed status variable to enumerated data type for
               clarity changed Send_Record, Receive_Record to
               independant procedures (callable by outside processes)

   15 Jun 89 - eliminated global variables, moved formal declarations
               for command packet building blocks into Interface
               section

   22 Jul 89 - added Respond_by_file

   28 Jul 89 - added a variable to control transfer monitor windows

   12 Aug 89 - extended variable Monitor_transfers to include the
               Update_status and the monitor window

   24 Aug 89 - gated ReadAux and WriteAux to show only data characters
               changed Respond_by_file to function to obtain status
```

```
                broke long resync problem with Command_Xfer syncing on
                CAN character from master and resetting after 10 block
                errors }

(********************  Start Edwards Excerpt  *********************)

CONST
      CEE = $43;

TYPE
  Result = ( Rx_sync,          { Waiting for sync           }
             Rx_done,          { completed                  }
             Rx_ACK,           { Good Rx, within retrymax   }
             Rx_old_ACK,       { Good Rx, old block         }
             Rx_EOT,           { Good Rx, EOT char          }
             Rx_junk,          { Garbage on the line        }
             Rx_timeout,       { nothing heard              }
             Rx_errors,        { Bad Rx, retrymax exceeded  }
             Rx_lost_block,    { Bad Rx, out of sync        }
             Rx_NAK,           { Bad Rx, NAK sent           }
             Rx_CAN,           { Good Rx, CAN char          }
             Rx_keypressed,    { Keypressed detected        }
             Tx_sync,          { Waiting for sync           }
             Tx_done,          { completed                  }
             Tx_ACK,           { Good Tx, within retrymax   }
             Tx_CEE_sync,      { Good Tx, CRC sync rxd      }
             Tx_EOT,           { Good Rx, EOT char          }
             Tx_timeout,       { nothing heard              }
             Tx_errors,        { Bad Tx, retrymax exceeded  }
             Tx_NAK_sync,      { Good Tx, cksum sync rxd    }
             Tx_NAK,           { Bad Tx, NAK received       }
             Tx_CAN,           { Bad Rx, CAN char received  }
             Tx_Junk,          { Trash on the receive line  }
             Tx_keypressed );{ Keypressed detected        }


VAR Suppress_EOT,
    Suppress_CAN,
    Monitor_Transfers : boolean;

FUNCTION Sync_Receive ( seconds : integer;
                        sync_character : char ) : result;



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

FUNCTION Receive_Record ( VAR Buf : Buffer; blocksize : word;
                              seconds : word; expected_block : word;
                          VAR errors : byte ) : result;

FUNCTION Sync_Send ( seconds : word ) : result;

FUNCTION Send_Record ( VAR Buf : Buffer; blocksize : word;
                           seconds : word; block : byte;
                         VAR errors : byte ) : result;
PROCEDURE Send_EOT ( VAR status : result );

PROCEDURE Send_CAN;

PROCEDURE Transfer_File (  Send : Boolean );

Function Command_Xfer(Send:Boolean; VAR buf : buffer;
                      BlockSize:Integer) : result;

FUNCTION Respond_by_file ( Response : pathstring ) : result;

Procedure Send_String ( S : String );

Function Get_response ( BlockSize:Integer ) : result;

Procedure string_to_buf ( s : string; VAR buf : buffer );

{ Converts a string into an Xmodem buffer }

Function buf_to_string ( VAR buf : buffer ) : string128;

{ Converts an Xmodem buffer into a string }

IMPLEMENTATION


CONST timeout = 256;
      Retrymax = 10;

TYPE
  Xmodem_Frame = ARRAY [1..4] of Char;



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Const Xmodem_Status:Array [Rx_sync..Tx_keypressed] of String[17] = (
            'Rx_sync          ', { Waiting for sync          }
            'Rx_done          ', { completed                 }
            'Rx_ACK           ', { Good Rx, within retrymax  }
            'Rx_old_ACK       ', { Good Rx, old block        }
            'Rx_EOT           ', { Good Rx, EOT char         }
            'Rx_junk          ', { Garbage on the line       }
            'Rx_timeout       ', { nothing heard             }
            'Rx_errors        ', { Bad Rx, retrymax exceeded }
            'Rx_lost_block    ', { Bad Rx, out of sync       }
            'Rx_NAK           ', { Bad Rx, NAK sent          }
            'Rx_CAN           ', { Good Rx, CAN char         }
            'Rx_keypressed    ', { Keypressed detected       }
            'Tx_sync          ', { Waiting for sync          }
            'Tx_done          ', { completed                 }
            'Tx_ACK           ', { Good Tx, within retrymax  }
            'Tx_CEE_sync      ', { Good Tx, CRC sync rxd     }
            'Tx_EOT           ', { Good Rx, EOT char         }
            'Tx_timeout       ', { nothing heard             }
            'Tx_errors        ', { Bad Tx, retrymax exceeded }
            'Tx_NAK_sync      ', { Good Tx, cksum sync rxd   }
            'Tx_NAK           ', { Bad Tx, NAK received      }
            'Tx_CAN           ', { Bad Rx, CAN char received }
            'Tx_Junk,         ', { Trash on the receive line }
            'Tx_keypressed    ');{ Keypressed detected       }


VAR
  CRC : Boolean;
  Xfer_File : File;
  Status_ID, Monitor_ID:Byte;
  Monitor_File:File of Char;
  buffr : buffer;
  monitor_gate : boolean;

PROCEDURE string_to_buf ( s : string; VAR buf : buffer );

{ Converts a string into an Xmodem buffer }

VAR index : byte;

BEGIN
  FOR index := 1 TO Length ( s ) DO
    buf [ index ] := s [ index ];

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

  FOR index := Length ( s ) + 1 TO 128 DO
    buf [ index ] := Char ( NUL );
END;


FUNCTION buf_to_string ( VAR buf : buffer ) : string128;

{ Converts an Xmodem buffer into a string }

CONST SPACE = ' ';
      TILDE = '~';

VAR s : string128;
    index : byte;

BEGIN
  s := '';
  FOR index := 1 TO 128 DO
    IF buf [ index ] IN [ SPACE .. TILDE ] THEN
      s := s + buf [index]
    ELSE s := s + SPACE;
  buf_to_string := s;
END;

FUNCTION ReadAux ( seconds : word ) : word;

VAR I : word;
    Ch : char;

BEGIN
  I:=seconds * 1000;
  While ((not RS232_Avail) and (I > 0) AND (NOT Keypressed)) do BEGIN
    Delay(1);
    DEC(I);
  End;
  If RS232_Avail then BEGIN
    Ch := RS232_In;
    If ( Monitor_ID > 0 ) AND ( monitor_gate ) then Begin
      TextColor(Foreground);
      TextBackground(Background);
      Case Byte(Ch) of
        NUL,BEL,BS,LF : { suppress };
        $20 .. $FF    : Write ( Ch );
        CR            : Writeln;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(*********************  Continue Edwards Excerpt  *********************)

      End;  {of Case}
      Write(Monitor_File,Ch);
    End;
    ReadAux := ORD(Ch);
    End
  else
    ReadAux := Timeout;
End;  {of ReadAux}

Procedure WriteAux(Ch:Char);

Begin
      RS232_Out(Ch);
      If ( Monitor_ID > 0 ) AND ( monitor_gate ) then
         Begin
         TextColor(Background);
         TextBackground(Foreground);
         Case Byte(Ch) of
           NUL,BEL,BS,LF : { suppress };
           $20 .. $FF    : Write ( Ch );
           CR            : Writeln;
         End;  {of Case}
         Write(Monitor_File,Ch);
End; {begin}
End;  {of WriteAux}

Procedure Send_String ( S : String );

VAR index : word;

BEGIN
  IF Length ( S ) > 0 THEN BEGIN
    FOR index := 1 TO Length ( S ) DO
      RS232_Out( S [ index ] );
    RS232_Out ( Char (CR) );
  END;

END;

FUNCTION Receive_Record ( VAR Buf : Buffer; blocksize : word;
                              seconds : word; expected_block : word;
                          VAR errors : byte ) : result;



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  *********************)

VAR
    temp : word;
    I : word;
    checksum : byte;
    Frame : Xmodem_frame;
    Ch : Char;

BEGIN
  Ch:=Char(NUL);
  errors := 0;

  CASE ReadAux ( seconds ) OF

    SOH  : BEGIN
             monitor_gate := false; { turn off monitor display }
             For I:=2 to 3 do
               Frame [I] := Char (Lo( ReadAux( seconds )));
             Checksum:=0;
             monitor_gate := true; { turn on monitor display }
             For I:=1 to BlockSize do
               Begin
                 Buf [I] := Char(Lo ( ReadAux (1)));
                 Checksum:= (Byte(Checksum)+Byte(Buf[I])) MOD 256;
               End;
             monitor_gate := false; { turn off monitor display }
             Frame [4] := Char(Lo(ReadAux( 1 )));
             If (Byte(Frame[2]) <> (255-Byte(Frame[3]))) or
               (Char(Checksum) <> Frame[4]) then
                 Begin     {Error on datacomm line}
                   INC(Errors);
                   WriteAux(Char(NAK));
                   Receive_Record := Rx_NAK;
                 End
             else if Byte(Frame[2]) = expected_block then
               Begin      {Block numbers match}
                 Errors:=0;
                 Receive_Record := Rx_ACK;
                 WriteAux(Char(ACK));
               End
             else if Byte(Frame[2]) = (expected_block-1) then begin
               Receive_Record := Rx_old_ACK;
               INC ( Errors );
               WriteAux(Char(ACK))  {Old block resent...ACK it}
               END

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

              else
                Begin   {We lost a block}
                  Receive_Record := Rx_lost_block;
                End;
            End; {SOH}

    CAN  :  Receive_record := Rx_CAN;

 Timeout :  Receive_record := Rx_timeout;

    EOT  :  Receive_record := Rx_EOT;

    else    Receive_record := Rx_junk;

      END; { OF CASE }

END; {Receive_Record}

PROCEDURE Get_Buffer ( VAR buf : buffer; blocksize : word );

VAR
  Numread : word;
  index : word;

BEGIN
  BlockRead(Xfer_File, buf, blocksize, Numread);
  IF Numread < blocksize THEN
  For index := Numread + 1 to blocksize DO
    Buf[index] := CHAR(ORD(0));
END;

FUNCTION Send_Record ( VAR Buf : Buffer; blocksize : word;
                           seconds : word; block : byte;
                       VAR errors : byte ) : result;


VAR
  Numread,
  Numwritten : word;
  irdex   : word;
  checksum : byte;
  Ch : CHAR;


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

  ending_char : char;
  quit : boolean;

BEGIN
        monitor_gate := false; { turn off monitor display }
        Errors := 0;
        checksum := 0;
        FOR index := 1 to blocksize DO
          checksum := (checksum + ORD ( Buf [index] )) MOD 256;
        Begin
               IF blocksize = 128 THEN WriteAux ( Char ( SOH ) )
                 ELSE WriteAux ( Char ( SOH ));;
               WriteAux ( Char ( Block ) );
               WriteAux ( Char ( 255-Block ) );
               monitor_gate := true; { turn on monitor display }
               For index :=1 to blocksize DO
                  WriteAux(Buf[index]);
               monitor_gate := false; { turn off monitor display }
               WriteAux ( Char ( checksum ));
               PurgeLine;
               CASE ReadAux ( seconds ) OF

                    ACK    :  Send_Record := Tx_ACK;

                    NAK     : Send_Record := Tx_NAK;

                    CAN     : Send_Record := Tx_CAN;

                    Timeout : Send_Record := Tx_timeout;

                    ELSE  Send_Record := Tx_Junk;

                 End; {case}

                 IF Keypressed THEN Send_Record := Tx_keypressed;
              END; {repeat}

End;

FUNCTION Sync_Receive ( seconds : integer;
                        sync_character : char ) : result;



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

VAR
  I,
  tries : integer;


BEGIN
  PurgeLine;
  WriteAux(sync_character); { try immediately }
  tries := TRUNC ( seconds /5 + 0.6 ); {convert seconds to cycles }
  WHILE ((not RS232_Avail) and ( tries > 0 )
          and ( NOT keypressed )) do BEGIN
    WriteAux(sync_character);
    I := 1000;
    While ((not RS232_Avail) and (I > 0 ) and ( NOT keypressed )) do
      Begin;
        Delay( 5 ); { 10 ms * 1000 cycles = 10 seconds }
        DEC ( I );
      End;
    DEC ( tries );
  END;
  IF Keypressed THEN
    Sync_Receive := Rx_keypressed
  ELSE IF RS232_Avail THEN BEGIN
         IF RS232_peek <> Char ( CAN ) THEN Sync_Receive := Rx_sync
         ELSE Sync_Receive := Rx_CAN;
       END
  ELSE Sync_Receive := Rx_timeout;
END;

FUNCTION Sync_Send ( seconds : word ) : result;

VAR
  quit : boolean;

Begin
      quit := FALSE;
      Repeat
        PurgeLine;




(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

        CASE ReadAux ( seconds ) OF
          CEE          : BEGIN { checksum handshake }
                           CRC := TRUE;
                           Sync_Send := Tx_CEE_sync;
                           quit := TRUE;
                         END;

          NAK          : BEGIN { checksum handshake }
                           CRC := FALSE;
                           Sync_Send := Tx_NAK_sync;
                           quit := TRUE;
                         END;

          Timeout      : BEGIN
                           Sync_Send := Tx_timeout;
                           quit := TRUE;
                         END;

          CAN          : BEGIN
                           Sync_send := Tx_CAN;
                           quit := true;
                         END;

          ELSE           BEGIN
                           Sync_send := Tx_junk; {Garbage on the line}
                         END;
        End; {CASE }
      UNTIL ( quit ) OR Keypressed;
      IF keypressed THEN Sync_Send := Tx_Keypressed;
END;

PROCEDURE Send_EOT ( VAR status : result );

VAR errors : byte;

  BEGIN
    IF ( Suppress_EOT ) THEN
      status := Tx_done
    ELSE BEGIN
      Errors := 0;
      REPEAT
        WriteAux ( Char ( EOT ));
        INC (Errors);
      UNTIL (ReadAux ( 10 ) = ORD ( ACK )) OR ( Errors = Retrymax );

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      IF  Errors = Retrymax THEN
        Status := Tx_timeout { timeout on EOT }
      ELSE status := Tx_done;
    END
  END;

PROCEDURE Send_CAN;

BEGIN
  IF NOT Suppress_CAN THEN BEGIN
    WriteAux ( char (CAN));
    WriteAux ( char (CAN));
  END;
END;

Function Xmodem_Xfer(Send:Boolean; BlockSize:Integer) : result;
{This procedure performs an Xmodem file transfer

 Input: Send - True to send a file
              False to receive a file

        BlockSize - The block size to use for the file transfer }
VAR ending_char : char;
    Xfer_Type:String[6];
    done,
    Abort:Boolean;
    Status : result;
    Ch : Char;
    Errors,
    Settings,
    Block_Count : byte;
    I,
    block,
    index,
    Blocks,
    Numread,
    Error_Count : word;
    Byte_Count: Longint;
    buf : buffer;

  Procedure Update_Status;
  Var I:Integer;



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

  Begin
    If Monitor_ID > 0 then begin
      If Get_Window(Status_ID) then;
      For I:=2 to 5 do Begin
        GotoXY(11,I);
        ClrEol;
        Case I of
          2:  Write(Xmodem_Status[Status]);
          3:  Write(Blocks);
          4:  Write(Byte_Count);
          5:  Write(Error_Count);
        End;  {of Case}
      End;
    If Get_Window(Monitor_ID) then;
    END;
  End;  {of Update_Status}


Begin
   If Monitor_Transfers THEN
      Begin
      If Open_Window(1,8,80,24,Flag_Borders,'Monitor Window') = 0 then;
      ClrScr;
      Writeln('Opening monitor file');
      Monitor_ID:=Active_Window^.ID;
      Assign(Monitor_File,'MONITOR.DAT');
      Rewrite(Monitor_File);
      End
   else
      Begin
      Monitor_ID:=0;
   End;
   Xfer_Type:='Xmodem';

   { Open the Status Window }

   If Open_Window(40,1,80,7,Flag_Borders,Xfer_Type) = 0 then;
   Status_ID:=Active_Window^.ID;
   ClrScr;
   For I:=1 to 5 do
      Begin
      GotoXY(1,I);



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      Case I of
         1:  Write('Name    : ');
         2:  Write('Status  :');
         3:  Write('Blocks  :');
         4:  Write('Bytes   :');
         5:  Write('Errors  :');
         End;  {of Case}
      End;
   RS_Eight_Bits; { make sure we can pass eight data bits }
   Blocks:=0;
   Byte_Count:=0;
   Errors:=0;
   Error_Count:=0;
   Block_Count:=1;
   Abort:=False;
   If Send then
      Begin   {Send the file}
      Status := Tx_sync; { Holding for start }
      Update_status;
      Status := Sync_Send ( 10 );

      If Status = Tx_Keypressed then Ch := ReadKey;
      Update_status;
      IF Status = Tx_Keypressed THEN
        { keep status same }
      ELSE IF NOT (Status IN [Tx_CEE_sync, Tx_NAK_sync]) THEN BEGIN
        Writeln ('No acknowledgement from other side');
        {Status := Tx_timeout;}
        Update_Status;
        END
      ELSE
        BEGIN
          done := false;
          While not (Eof(Xfer_File)) AND NOT (done) do
           Begin
           Update_Status;
           Get_Buffer ( buf, blocksize );
           status := Send_Record ( Buf, Blocksize, 10, block_count,
                                   errors );




(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

          CASE Status OF
            Tx_ACK : BEGIN
                       Error_Count := Error_Count + Errors;
                       INC(Blocks);
                       Byte_Count:=Byte_Count+BlockSize;
                       INC(Block_Count);
                     END;

            Tx_NAK : BEGIN
                       INC(Error_count);
                       If Error_count >= retrymax then done := true;
                     END;

            TX_timeout : BEGIN
                           INC(Error_count);
                           If Error_count >= retrymax then
                             done := true;
                         End;

            Tx_CAN,
            Tx_keypressed : BEGIN
                              done := TRUE;
                            END;

            ELSE      BEGIN
                        INC(Error_count);
                        If Error_count >= retrymax then
                          Begin
                          done := true;
                          Status := Tx_errors;
                          End;
                      END;
          END;
          Update_Status;
        End; {WHILE}
       While KeyPressed do
          Begin
          Ch := ReadKey;
          End;
      END;
     If Status = TX_ACK then Send_EOT ( status )
     ELSE Send_CAN;
    END


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

   else
      Begin   {Receive file}
      Status:=Rx_sync;
      Update_status;
      Status := Sync_Receive ( 60, Char(NAK) );
      CASE Status OF
        Rx_KeyPressed   : Begin
                            Abort := TRUE;
                            Update_status;
                            Ch := ReadKey;
                          End;

        Rx_timeout,
        Rx_CAN          : BEGIN
                            Abort := TRUE;
                            Update_Status;
                          END;

        ELSE              Repeat
         Status := Receive_Record ( Buf, blocksize, 1,
                                    Block_count, errors );
         CASE Status OF
           Rx_ACK : BEGIN
                      INC(Blocks);
                      Byte_Count:=Byte_Count+BlockSize;
                      INC ( Block_Count );
                      BlockWrite(Xfer_File,Buf, blocksize );
                    END;
           Rx_junk,
           Rx_timeout,
           Rx_Old_ACK : BEGIN
                         INC ( Error_Count );
                         IF Error_Count > retrymax THEN
                            abort := TRUE;
                        END;

           Rx_EOT : BEGIN
                      Status := Rx_EOT;
                    END;
           ELSE BEGIN
                Error_Count := Error_Count + Errors;
                IF Error_Count > retrymax THEN abort := TRUE;
           END;
         END; {CASE}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

         Update_Status;
         If not Abort then
           While KeyPressed do
            Begin
            Ch := ReadKey;
            Abort:=True;
            Status:=Rx_keypressed;
            End;
        Until (Status = Rx_EOT ) or Abort;
      END; { CASE }
      If not Abort then Status:=Rx_done;
      Update_Status;
      If Status <> Rx_done then
         WriteAux(Char(CAN))
      else
         WriteAux(Char(ACK));
   End;
   Xmodem_Xfer := status;
   Close(Xfer_File);
   If (not Send) and (Abort) then
      Erase(Xfer_File);

   { Close the Status window }
   RS_Restore ( Current_COM ); { restore comport settings to whatever
                                 was selected before }
   If Close_Window then;
   If Monitor_ID > 0 then
      Begin
      If Close_Window then; { Close the monitor window if open }
      Textcolor(Foreground);
      Textbackground(Background);
      Close(Monitor_File);
      Monitor_ID := 0;
      End;
End;  {of Xmodem_Xfer}


Function Command_Xfer(Send:Boolean; VAR buf : buffer;
                      BlockSize:Integer) : result;

{This procedure performs an command/response exchange

 Input: Send - True to send a buffer
              False to receive buffer

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

        BlockSize - The block size to use for the transfer
}

VAR ending_char : char;
    Xfer_Type:String[18];
    done,
    Abort:Boolean;
    Status : result;
    Ch : Char;
    Errors,
    Settings,
    Block_Count : byte;
    I,
    index,
    Blocks,
    Numread,
    Error_Count : word;
    Byte_Count: Longint;

  Procedure Update_Status;
  Var I:Integer;
  Begin
    If Monitor_ID > 0 then begin
      If Get_Window(Status_ID) then;
      For I:=2 to 5 do Begin
        GotoXY(11,I);
        ClrEol;
        Case I of
          2:  Write(Xmodem_Status[Status]);
          3:  Write(Blocks);
          4:  Write(Byte_Count);
          5:  Write(Error_Count);
        End;  {of Case}
      End;
    If Get_Window(Monitor_ID) then;
    END;
  End;  {of Update_Status}


Begin
   If Monitor_Transfers THEN
   Begin
      If Open_Window(1,8,80,24,Flag_Borders,'Monitor Window') = 0 then;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      ClrScr;
      Writeln('Opening monitor file');
      Monitor_ID:=Active_Window^.ID;
      Assign(Monitor_File,'MONITOR.DAT');
      Rewrite(Monitor_File);
      Xfer_Type:='Command Transfer';
      If Open_Window(40,1,80,7,Flag_Borders,Xfer_Type) = 0 then;
      Status_ID:=Active_Window^.ID;
      ClrScr;
      For I:=1 to 5 do Begin
        GotoXY(1,I);
        Case I of
          1:  Write('');
          2:  Write('Status  :');
          3:  Write('Blocks  :');
          4:  Write('Bytes   :');
          5:  Write('Errors  :');
        End;  {of Case}
    End;
  End
  else
     Monitor_ID:=0;
  RS_Eight_Bits; { make sure we can pass eight data bits }
  Blocks := 0;
  Errors := 0;
  Byte_Count:=0;
  Error_Count:=0;
  Block_Count:=1;
  Abort:=False;
  If Send then
     Begin   {Send the command}
     Status := Tx_sync; { Holding for start }
     Update_status;
     Status := Sync_Send ( 10 );

     If Status = Tx_Keypressed then Ch := ReadKey;
     Update_status;
     IF Status = Tx_Keypressed THEN
       { keep status same }
     ELSE IF NOT (Status IN [Tx_CEE_sync, Tx_NAK_sync]) THEN BEGIN
       Writeln ('No acknowledgement from other side');
       {Status := Tx_timeout;}
       Update_Status;
       END

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      ELSE
        BEGIN
          done := false;
          REPEAT
           Update_Status;
           status := Send_Record ( Buf, Blocksize, 10, block_count,
                                        errors );
           CASE Status OF

             Tx_ACK : BEGIN
                        Error_Count := Error_Count + Errors;
                        Byte_Count:=Byte_Count+BlockSize;
                        done := true;
                      END;

             Tx_NAK : BEGIN
                        INC(Error_count);
                        If Error_count >= retrymax then done := true;
                      END;

             TX_timeout : BEGIN
                            INC(Error_count);
                            If Error_count >= retrymax then
                               done := true;
                          End;

             Tx_CAN,
             Tx_keypressed : BEGIN
                               Writeln('aborting');
                               done := TRUE;
                             END;

             ELSE     BEGIN
                        INC(Error_count);
                        If Error_count >= retrymax then
                          Begin
                          done := true;
                          Status := Tx_errors;
                          End;
                      END;
           END; {Case }
```

```
(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

          UNTIL done;
          While KeyPressed do
            Begin
            Ch := ReadKey;
            End;
        END;
      Update_status;
      If Status = TX_ACK then BEGIN
        Send_EOT ( status );
        status := Tx_done;
      END
      ELSE Send_CAN;
    END
  else
    Begin   {Receive file}
      Status:=Rx_sync;
      Update_status;
      Status := Sync_Receive ( 60, Char(NAK) );
      CASE Status OF
        Rx_KeyPressed : BEGIN
                          Abort := TRUE;
                          Update_status;
                          Ch := ReadKey;
                        End;

        Rx_timeout,
        Rx_CAN        : BEGIN
                          Abort := TRUE;
                          Update_Status;
                        END;

        ELSE Repeat
         Status := Receive_Record ( Buf, blocksize, 10,
                                    Block_count, errors );
         CASE Status OF
           Rx_ACK     : BEGIN
                          Byte_Count:=Byte_Count+BlockSize;
                        END;
           Rx_junk,
           Rx_timeout,
           Rx_Old_ACK : BEGIN
                         INC ( Error_Count );



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                          IF Error_Count > retrymax THEN
                            abort := TRUE;
                          END;

          Rx_EOT      : BEGIN
                          Status := Rx_EOT;
                        END;

          ELSE          BEGIN
                          Error_Count := Error_Count + Errors;
                          IF Error_Count > retrymax THEN
                            abort := TRUE;
                        END;
        END; {CASE}
        Update_Status;
        If not Abort then
          While KeyPressed do
           Begin
           Ch := ReadKey;
           Abort:=True;
           Status:=Rx_keypressed;
           End;
        Until (Status = Rx_EOT ) or Abort;
     END; { CASE }
     If not Abort then Status:=Rx_done;
     Update_Status;
     If Status <> Rx_done then
        WriteAux(Char(CAN))
     else
        WriteAux(Char(ACK));
  End;  { Receive }
  Command_Xfer := status;

  { Close the status window }
  { restore comport settings to whatever was selected before }
  RS_Restore ( Current_COM );
  If Monitor_ID > 0 then { Close the monitor window }
     Begin
     If Close_Window then;
     If Close_Window then;
     Textcolor(Foreground);
     Textbackground(Background);
     Close(Monitor_File);
     Monitor_ID := 0;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      End;
End;  {of Command_Xfer}


Procedure Transfer_File(Send:Boolean);
{This procedure initiates a file transfer

 Input: Send - True if we want to send a file
               False to receive a file
}
Var FileName : Long_String;
    I,J:Integer;
    Abort:Boolean;
    status : result;

Begin
   Abort:=False;
   If Open_Window(20,16,60,19,Flag_Borders,'Name') = 0 then;
   ClrScr;
   Write('File Name: ');
   Readln( FileName );
   If Close_Window then;
   IF ( Length ( FileName ) = 0 ) or (FileName = ^P) THEN
     { do nothing }
   ELSE BEGIN
     For J:=1 to Length( FileName ) do
       FileName [J] := UpCase( FileName [J] );
     Assign ( Xfer_File,   FileName );
     If Send then Begin
{$I-}
        Reset(Xfer_File, 1);
{$I+}
        If IOResult > 0 then Begin
          NoFile( FileName );
          Abort:=True;
        End;
     End
     else
       Rewrite(Xfer_File, 1);
   If not Abort then  status := Xmodem_Xfer ( Send, 128 );
  END;
End;  {of Transfer_File}


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

FUNCTION Respond_by_file ( Response : pathstring ) : result;

{This procedure provides the user a response contained in a file

 Input: Response - the complete path specification for the file
}

CONST Send : boolean = TRUE;

Var
    Abort:Boolean;

Begin
   Abort:=False;
   Assign ( Xfer_File, Response );
{$I-}
   Reset(Xfer_File, 1);
{$I+}
   If IOResult > 0 then
     Begin
     NoFile(Response);
     Abort:=True;
     End;
   If not Abort then  Respond_by_file := Xmodem_Xfer ( Send, 128 )
   ELSE Respond_by_file := Tx_CAN;
End;  { Respond_by_file }

Function Get_response ( BlockSize:Integer ) : result;

{This procedure performs an Xmodem file transfer

 Input: Send - True to send a response
              False to receive a series of responses

        BlockSize - The block size to use for the file transfer

        Status_ID, Monitor_ID must be seen by WriteAux, ReadAux
}

VAR ending_char : char;
    Xfer_Type:String[6];
    done,
    Abort:Boolean;
    Status : result;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

    Ch : Char;
    Errors,
    Settings,
    Block_Count : byte;
    I,
    block,
    index,
    Blocks,
    Numread,
    Error_Count : word;
    Byte_Count: Longint;
    buf : buffer;
    Display_Window_ID : byte;

Begin
   Monitor_ID := Active_Window^.ID;
   Assign ( Monitor_File, 'NUL' );
   Rewrite ( Monitor_File );
   { Change to current comms }
   RS_Eight_Bits; { make sure we can pass eight data bits }
   Blocks:=0;
   Byte_Count:=0;
   Errors:=0;
   Error_Count:=0;
   Block_Count:=1;
   Abort:=False;
   Begin   {Receive file}
      Status:=Rx_sync;
      Status := Sync_Receive ( 60, Char(NAK) );
      CASE Status OF
        Rx_KeyPressed   : Begin
                            Abort := TRUE;
                            Ch := ReadKey;
                          End;

        Rx_timeout,
        Rx_CAN          : Abort := TRUE;

      ELSE Repeat
         Begin
         Status := Receive_Record ( Buf, blocksize, 1,
           Block_count, errors );


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

        CASE Status OF
          Rx_ACK : BEGIN
                     INC(Blocks);
                     Byte_Count:=Byte_Count+BlockSize;
                     INC ( Block_Count );
                   END;
          Rx_junk,
          Rx_timeout,
          Rx_Old_ACK : BEGIN
                        INC ( Error_Count );
                        IF Error_Count > retrymax THEN abort := TRUE;
                       END;

          Rx_EOT : BEGIN
                     Status := Rx_EOT;
                   END;

          Rx_CAN : BEGIN
                     abort := TRUE;
                   END;

          ELSE BEGIN
               Error_Count := Error_Count + Errors;
               IF Error_Count > retrymax THEN abort := TRUE;
          END;
        END; {CASE}
        If not Abort then
          While KeyPressed do
           Begin
           Ch := ReadKey;
           Abort:=True;
           Status:=Rx_keypressed;
           End;
      END { Receive }
    Until (Status = Rx_EOT ) or Abort;
  END; { CASE }
  If not Abort then Status:=Rx_done;
  If Status <> Rx_done then
       WriteAux(Char(CAN))
  else
       WriteAux(Char(ACK));
End;
Get_Response := status;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

   { restore comport settings to whatever was selected before }
   RS_Restore ( Current_COM );

   If Monitor_ID > 0 then
      Begin
      Textcolor(Foreground);
      Textbackground(Background);
      Close(Monitor_File);
      Monitor_ID := 0;
    End;
End;  { Get_response }


BEGIN
  Suppress_EOT := FALSE;
  Suppress_CAN := FALSE;
  Monitor_Transfers := TRUE;
  monitor_gate := false; { don't display xmodem packet headers }
END.
```

```
(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*********************  End Edwards Excerpt  *********************)
```

{

APPENDIX AA

SOURCE LISTING FOR PROGRAM DISTRIB

```
}
(*************************************************************************)
(****                        DISTRIB.PAS                              ****)
(****  This is the main program for the Master/Slave networked        ****)
(****  computer system.  The same program is used for both the        ****)
(****  Master and Slave, with the function being selected from        ****)
(****  the command line as follows:                                   ****)
(****                                                                 ****)
(****  Master:  Distrib Master (also allows program config            ****)
(****                                                                 ****)
(****  Server:  Distrib Server                                        ****)
(****                                                                 ****)
(****  Reference:  Edwards, C. C., Advanced Techniques in Turbo       ****)
(****              Pascal, pp. 220-275, Sybex, Inc., 1987             ****)
(****                                                                 ****)
(****  Heavily modified from the terminal emulation program           ****)
(****  found in the reference.  Converted to a Turbo Pascal 4.0       ****)
(****  program by Nelson Ard                                          ****)
(****                                                                 ****)
(****  Last Modification:  Sep 89                                     ****)
(*************************************************************************)

(*  Modification history
    12 Sep 89 - Replaced local RS232 write procedure with
                DataCom.Send_String
*)

{$R+}    {Range checking on}
{$B+}    {Boolean complete evaluation on}
{$S+}    {Stack checking on}
{$I+}    {I/O checking on}
{$N-}    {No numeric coprocessor}
{$M 65500,16384,65500} {Modified default stack and heap}

Program Distrib;

Uses
  Datacom,
  Crt,
  Dos,
  Wndow,
  Xmodm,
  Director,
  General,
```

```
  ErrorCod,
  Support,
  Printer,
  Parser, Spawn, miscpack;

(********************  Start Edwards Excerpt  ********************)

Procedure Save_File(D:Boolean);
{This procedure asks the user if he wants to save a changed
 configuration If so, it writes the appropriate file

 Input D: True if saving default values
          False if saving phone file
}
Var Configure:File of Byte;
    Phone:Phone_Record;
    J:Integer;

   Begin
   If Open_Window(50,9,67,12,Flag_Borders,'') = 0 then;
   ClrScr;
   If D then
      Write('Save defaults?')
   else
      Write('Save this entry?');
   If Yes('Save') then
     Begin
     ClrScr;
     Write('Saving...');
     If D then
        Begin
        Assign(Configure,Defaults.Default_Name);
{$I-}
        Reset(Configure);
{$I+}
        If IOResult > 0 then
           NoFile(Defaults.Default_Name)
        else
           Begin
           ClrScr;
           Writeln('If you want to use these parameters');
           Write('You must end and restart Distrib');
           OK('');
           If Close_Window then;
           End;
        End

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

     else
        Begin
{$R-}
        Assign(Phone_File,'DISTRIB.PHN');
        Rewrite(Phone_File);
        For J:=1 to Phone_Menu^.Length do
           Begin
           Phone.Name:=Phone_Menu^.Names[J];
           Phone.Phone_Data:=Phone_Stuff^[J];
           Write(Phone_File,Phone);
           End;
        Close(Phone_File);
{$R+}
        End;
     End;
   If Close_Window then;
   End;  {of Save_File}

{$V-}
Procedure Write_AUX_String ( S : STRING );
{This procedure writes a string out to the currently selected COM port}
VAR index : byte;

BEGIN
  FOR index := 1 TO Length(S) DO BEGIN
    RS232_Out(S[index]);
  END;
END;
{$V+}

Procedure Dial_Phone(I:Integer; Demon_Dial:Boolean);
{This procedure dials a phone entry.  The demon dial feature is the
 only feature of Distrib which explicitly assumes the presense of
 a Hayes or Hayes compatible modem.

 Input: I - The index into the phone array that we are to dial
        Demon_Dial - true if we are to repetitively dial until an
                     answer is obtained
}
Var Count:Integer;
    S:Long_String;
    Ch:Char;
    {Connected:Boolean;}


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

    J:Integer;
    Time:Integer;
    quit : boolean;

    Procedure Flush_Buffer;
       Var Ch:Char;
       Begin
       Repeat Begin
              Ch := RS232_In;
              If not RS232_Avail then Delay(200);
              End
       Until not RS232_Avail;
       End;  {of Flush_Buffer}

   Begin
   RS_Cleanup;
{$R-}
   With Phone_Stuff^[I] do
      Begin
      RS_Initialize(Defaults.Default_Modem,Phone_Baud,Phone_Parity,
                    Phone_Stop,Phone_Length);
      Echo:=Phone_Echo;
      End;
   Last_Dial:=I;
   If Demon_Dial then
      Begin
      DataCom.Send_String('ATZ'+Char(CR));
      Flush_Buffer;
      Delay(1000);   {Give modem time to reset}
      DataCom.Send_String('ATV1Q0E1S7='+Char(Dial_Delay)+Char(CR));
      If Open_Window(15,09,65,17,Flag_Borders,'Dial') = 0 then:
      ClrScr;
      Writeln('Name      :',Phone_Menu^.Names[I]);
      Writeln('Attempt   :');
      Writeln('Status    :');
      Writeln('Started   :',Get_Time);
      Writeln('Dialed at :');
      Writeln('Elapsed   :');
      Write('Options   :ESC to abort...any other key to cycle');
      Flush_Buffer;
      Count:=0;
      quit := False;


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      Repeat Begin
             Count:=Count+1;
             Time:=0;
             GotoXY(12,2);
             Write(Count);
             GotoXY(12,3);
             ClrEol;
             Write('Dialing');
             GotoXY(12,5);
             Write(Get_Time);

DataCom.Send_String(Phone_Prefix+Phone_Stuff^[I].Phone_Number
                              +Char(CR));
             Flush_Buffer;
             J:=0;
             Delay(2000);   {Give time to dial the phone}
             While not (KeyPressed or RS232_Avail) do
                Begin
                Delay(10);   {This delay is correct for the PC or XT,
                              it may have to be changed for an AT or
                              faster box}
                J:=J+1;
                If J = 100 then
                   Begin
                   Time:=Time+1;
                   GotoXY(12,6);
                   ClrEol;
                   Write(Time,' Seconds');
                   J:=0;
                   End;
                End;
             If KeyPressed then
                Begin
                Ch := ReadKey;
                If KeyPressed then
                   Ch := ReadKey;
                If Ch = Char(ESC) then
                   Begin
                   S:='Aborted';
                   quit := True;
                   End
                else
                   S:='Cycling';


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                DataCom.Send_String(Char(Ch));
                Delay(2000);
                If RS232_Avail then Flush_Buffer;
                End
             else
                Begin
                S:='';
                Repeat Ch := RS232_In until Ch = Char(LF);
                Repeat Begin
                       Ch := RS232_In;
                       If Ch > Char(US) then
                          S:=S+Ch;
                       End
                until Ch = Char(LF);
                End;
             GotoXY(12,3);
             ClrEol;
             Write(S);
             If not Connected then Delay(5000);
             End
       Until Connected OR quit;
       For Count:=1 to 10 do Beep(500);
       If Close_Window then;
       End
    else
       DataCom.Send_String(Phone_Prefix+Phone_Stuff^[I].Phone_Number+
                           Char(CR));
{$R+}
    End;  {of Dial_Phone}

Procedure Dialing_Directory;
{This procedure allows the user to dial or modify any of the entries in
 the phone array}
Var I,J:Integer;

    Function Get_Dial:Integer;
       Begin
       If Open_Window(24,5,56,Min(6+Phone_Menu^.Length,17),Flag_Borders,
                      'Phone List') = 0 then;
       Get_Dial:=Process_Window_Menu(Phone_Menu^);
       If Close_Window then;
       End;  {of Get_Dial}



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

   Begin
   If Open_Window(36,5,44,11,Flag_Borders,'Choice') = 0 then;
   I:=Process_Window_Menu(Dial_Menu);
   If Close_Window then;
   Case I of
      0:  ; {ESC...No Choice}
      1,2:Begin  {Dial or Demon Dial}
          J:=Get_Dial;
          If J > 0 then
             Dial_Phone(J,I=2);
          End;
      3:  Begin  {Modify}
          I:=Get_Dial;
          If I > 0 then
             Modify_Entry(I);
          End;
      4:  Begin {Delete}
          If Phone_Menu^.Length = 1 then
             Begin
             If Open_Window(45,9,67,12,Flag_Borders,'') = 0 then;
             ClrScr;
             Write('Cannot delete last entry');
             OK('');
             If Close_Window then;
             End
          else
             Begin
             I:=Get_Dial;
             If I > 0 then
                Begin
                Old_Phone_Menu:=Phone_Menu;
                Old_Phone_Stuff:=Phone_Stuff;
                J:=Phone_Menu^.Length;
                GetMem(Phone_Stuff,(J-1)*Sizeof(Phone_Params));
                GetMem(Phone_Menu,(J-1)*Sizeof(Phone_Name)+2);
                Move(Old_Phone_Menu^,Phone_Menu^,(I-1)*
                     Sizeof(Phone_Name)+2);
                Move(Old_Phone_Stuff^,Phone_Stuff^,(I-1)*
                     Sizeof(Phone_Params));
                If I < J then
                   Begin
                   Move(Old_Phone_Menu^.Names[I+1],
                        Phone_Menu^.Names[I],
                        (J-I)*Sizeof(Phone_Name));

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                    Move(Old_Phone_Stuff^[I+1],Phone_Stuff^[I],
                         (J-I)*Sizeof(Phone_Params));
                    End;
                 Phone_Menu^.Length:=J-1;
                 FreeMem(Old_Phone_Menu,J*Sizeof(Phone_Name)+2);
                 FreeMem(Old_Phone_Stuff,J*Sizeof(Phone_Params));
                 Save_File(False);
                 End;
              End;
           End;
        5: Begin   {Add}
{$R-}
           Old_Phone_Menu:=Phone_Menu;
           Old_Phone_Stuff:=Phone_Stuff;
           GetMem(Phone_Stuff,(Phone_Menu^.Length+1)*
                  Sizeof(Phone_Params));
           GetMem(Phone_Menu,(Phone_Menu^.Length+1)*
                  Sizeof(Phone_Name)+2);
           I:=Old_Phone_Menu^.Length;
           Move(Old_Phone_Menu^,Phone_Menu^,I*Sizeof(Phone_Name)+2);
           Move(Old_Phone_Stuff^,Phone_Stuff^,I*Sizeof(Phone_Params));
           I:=I+1;
           Phone_Menu^.Length:=I;
           Phone_Menu^.Names[I]:='...To be provided...          ';
           Move(Defaults.Default_Phone,Phone_Stuff^[I],
                Sizeof(Phone_Params));
           Modify_Entry(I);
           FreeMem(Old_Phone_Menu,(I-1)*Sizeof(Phone_Name)+2);
           FreeMem(Old_Phone_Stuff,(I-1)*Sizeof(Phone_Params));
{$R+}
           End;
        End;  {of Case}
     End;  {of Dialing_Directory}

Procedure Dirs;

{ Replacement directory }

CONST
  Start  : integer = 5;
  Finish : integer = 20;



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

VAR
  error : integer;
  DirInfo : Dos.SearchRec;
  S,
  Mask,
  Option    : string;
  Directory_found : Boolean;
  FromLine : integer;
  Ch : Char;

Begin
  GetDir(0,S);
  If Open_Window(1,Start,80,Finish,Flag_Borders,S) = 0 then;
  ClrScr;
  IF Open_Window ( 5, Start + 5, 70, Start + 7, Flag_Borders,
                   'Mask? *.* is default' ) = 0 THEN;
  GotoXY ( 1,1 );
  Readln ( Mask );
  IF Length (Mask ) = 0 THEN Mask := '*.*';
  If Close_Window then;
  ClrScr;
  IF Open_Window ( 5, Start + 5, 70, Start + 7, Flag_Borders,
                   'Options?' ) = 0 THEN;
  GotoXY ( 1,1 );
  Write ('[ none = dir (Mask), ''w'' =  dir (Mask) /w ] ');
  Readln ( Option );
  If Close_Window then;
  ClrScr;
  IF Length ( Option ) = 0 THEN BEGIN
    GotoXY ( 1,1 );
    ShowDir ( Mask, 1, 13, error );
  END
  ELSE CASE Option[1] of
    'w', 'W'  : BEGIN
                  GotoXY ( 1,1 );
                  ViewDir (Mask, 1, 13 );
                END;
  END; {CASE}
  GotoXY ( 1, 13 );
  Write('Finished...Press any key');
  Ch := ReadKey;
  If KeyPressed then Ch := ReadKey;
  If Close_Window then;
End;  {of Dirs}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)


Procedure Change_DC_Parameters;
{This procedure allows the user to choose from a list of speed,
 parity, word length, and stop bit configurations}
Var I:Integer;
   Begin
   If Open_Window(67,1,79,23,Flag_Borders,'Baud-P-L-S') = 0 then;
   ClrScr;
   I:=Process_Window_Menu(Communications_Menu);
   If I > 0 then
      Begin
      RS_Cleanup;
      With Communications_Stuff[I] do
         Begin
         RS_Initialize(Current_Com,Speed,Parity,Stop,Length);
         End;
      End;
   If Close_Window then;
   End;  {of Change_DC_Parameters}

Procedure Hangup;
{This procedure hangs up the Hayes compatible modem}
Var Ch:Char;
   Begin
   Repeat Begin
          While RS232_Avail do Ch := RS232_In;
          Delay(500);
          End
   Until not RS232_Avail;
   DataCom.Send_String('+++');
   Delay(2500);
   DataCom.Send_String('ATH0'+Char(CR));
   Delay(1000);
   While RS232_Avail do Ch := RS232_In:
   End;  {of Hangup}

Procedure Dos_Shell;
{This procedure opens a window and spawns a DOS command processor}
Var Prog,Param,Dir:String;
I:Integer;




(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Begin
   Prog:=Find_Environment('COMSPEC');
   If Length(Prog) <> 0 then BEGIN
     Param:=' ';
     If Open_Window(40,5,60,8,Flag_Borders,'DOS') = 0 then;
     ClrScr;
     Writeln('Opening Dos Shell');
     Write('Use EXIT when done');
     OK('');
     If Close_Window then;
     If Open_Window(1,1,80,25,0,'') = 0 then;
     ClrScr;
     GetDir(0,Dir);
     Exec (Prog, Param);
     System.ChDir(Dir);
     if doserror <> 0 THEN BEGIN
       If Open_Window(40,1,75,3,Flag_Borders,'DOS Error') = 0 then;
       ClrEol;
       Writeln (Error_Code[DosError]);
       Delay ( 2000 );
       If Close_Window then;
     END;
   If Close_Window then;
   END
   ELSE BEGIN
     If Open_Window(35,10,75,13,Flag_Borders,'Error') = 0 then;
     ClrEol;
     Writeln(' Unable to open DOS shell');
     Write(' ''COMSPEC'' not found in environment');
     OK('');
     If Close_Window then;
   END;
End;  (of Dos_Shell)


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*********************  End Edwards Excerpt  *********************)

FUNCTION Operator_input ( Title : Window_title;
                          Prompt : PathString ) : PathString;

VAR Response : PathString;

BEGIN
    IF Open_Window ( 5, 10, 75, 13, Flag_Borders, Title ) = 0 THEN
    BEGIN
      ClrScr;
```

```
      Writeln ( Prompt );
      Readln ( Response );
      Operator_Input := Response;
      IF Close_Window THEN;
    END
    ELSE Operator_Input := '';
END;

PROCEDURE Operator_message ( Title : Window_title;
                             Message : PathString );


BEGIN
  IF Open_Window ( 40, 10, 80, 13, Flag_Borders, Title ) = 0 THEN BEGIN
      ClrScr;
      Writeln ( Message );
  END
END;

FUNCTION Process_command : result;

CONST Receive  : boolean = FALSE;
      Transmit : boolean = TRUE;

VAR
  index : byte;
  Response  : String128;
  Restype   : Response_type;
  Error_msg : String128;
  Errtype   : Response_type;
  Prompt    : String128;
  buf : buffer;
  send : boolean;
  Server_ID : byte;
  status : result;
  Ch : char;
  finished : boolean;
  debugging : boolean;

BEGIN
  debugging := FALSE;
  finished := FALSE;
  IF Open_Window ( 1, 1, 80, 7, Flag_Borders, 'Remote Server') = 0 THEN;
  ClrScr;
  Server_ID := Active_Window^.ID;
  For index := 1 TO 4 do BEGIN
    GotoXY ( 1, index );
    CASE index OF
      1 : Write ('Server Version 1.0');
      2 : Write ('Function : Initializing');
      3 : Write ('Status   : Awaiting Command');
```

```
    4 : Write ('Command  : ');
  END; { CASE }
END;

Send := FALSE;
Redirection := true;
{Send_string ('xmodem st test.tst');}
IF Get_Window ( Server_ID ) THEN;
GOTOXY ( 12, 2 );
Write ('Getting Command');
REPEAT
  status := Command_Xfer ( Receive, buf, 128 );
UNTIL ( status = Rx_done ) OR ( status = Rx_keypressed );
IF Get_Window ( Server_ID ) THEN;
Process_command := status;
IF ( status = Rx_keypressed ) AND NOT ( debugging ) THEN BEGIN
  IF Close_window THEN;
  WHILE keypressed DO
    Ch := readkey;
  EXIT;
END;
GOTOXY ( 12, 2 );
Write ('Parsing Command');
GOTOXY ( 12, 3 );
Write ('Executing Command ');
GOTOXY ( 12, 4 );

IF debugging THEN
  String_to_buf ( Operator_Input ( 'Command', 'server command?'),
                  buf );

Write ( buf_to_string ( buf ) );

Parser_main( buf_to_string ( buf), Response, Restype,
             Error_msg, Errtype, Prompt );

CASE Errtype OF

    strng     : BEGIN
                  IF Length ( Error_msg ) > 0 THEN BEGIN
                    string_to_buf ( Error_msg, buf );
                    REPEAT
                      status := Command_Xfer ( Transmit, buf, 128 );
                    UNTIL ( status = Tx_done )
                       OR ( status = Tx_keypressed )
                       OR ( status = Tx_CAN );
                    Process_command := status;
                    CASE status OF
```

```
                    Tx_keypressed  : BEGIN
                                       IF Close_window THEN;
                                       WHILE keypressed DO
                                         Ch := readkey;
                                       finished := TRUE;
                                       EXIT;
                                     END;

                    Tx_CAN         : BEGIN
                                       finished := TRUE;
                                     END;
                  END; {CASE}
                END;   {IF}
              END;

  file_type : BEGIN
                status := Xmodm.Respond_by_file ( Error_msg );
              END;

  nothing   : BEGIN
              END;

END; {CASE}

IF NOT (( finished ) OR ( status = Tx_CAN )) THEN

  CASE Restype OF

  strng     : BEGIN
                string_to_buf ( Response, buf );
                REPEAT
                  status := Command_Xfer ( Transmit, buf, 128 );
                UNTIL ( status = Tx_done )
                   OR ( status = Tx_keypressed )
                   OR ( status = Tx_CAN );
                Process_command := status;
                  CASE status OF

                    Tx_keypressed  : BEGIN
                                       IF Close_window THEN;
                                       WHILE keypressed DO
                                         Ch := readkey;
                                       finished := TRUE;
                                       EXIT;
                                     END;

                    Tx_CAN         : BEGIN
                                       finished := TRUE;
                                     END;
                  END; {CASE}
              END;
```

```
      file_type : BEGIN
                    status := Xmodm.Respond_by_file ( Response );
                  END;

      nothing   : BEGIN
                  END;

    END; { CASE }
    IF NOT finished THEN BEGIN
      IF Get_Window ( Server_ID ) THEN;

      GOTOXY ( 12, 2 );
      Write ('Forwarding Prompt');
      GOTOXY ( 12, 3 );
      Write ('Command Complete');
      GOTOXY ( 1, 4 );
      Write ('Prompt  : ' );
      GOTOXY ( 1, 11 );
      Write ( Prompt );
      string_to_buf ( ^M + Prompt, buf );
      REPEAT
        status := Command_Xfer ( Transmit, buf, 128 );
      UNTIL  ( status = Tx_done )
          OR ( status = Tx_keypressed )
          OR ( status = Tx_CAN );
      Send_CAN;
      WHILE keypressed DO
        Ch := readkey;
      Process_command := status;
      IF Get_Window ( Server_ID ) THEN;
    END;
    IF Close_window THEN;
END;

(************************  Start Edwards Excerpt  *********************)
{  1 Sep 89 global variables eliminated }

CONST Comms_Menu : integer = 9;
      Comms_Fns  : ARRAY [1..9] OF STRING [ 24 ] = (
                   'Initialize port         ',
                   'Connect to current port ',
                   'Disconnect current port ',
                   'ZCOPY file to remote    ',
                   'ZCOPY file from remote  ',


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
*********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                    'Get machine status      ',
                    'Login to remote machine ',
                    'Reset remote server     ',
                    '(ESC) Exit              ');
      Comms_Stat_Menu : integer = 7;
      Comms_Stat : ARRAY [ 1..7 ] OF STRING [ 16 ] = (
                    'Comm Port       ',
                    'Speed           ',
                    'Word Length     ',
                    'Parity          ',
                    'Stop Bits       ',
                    'Function        ',
                    'Status          ');


FUNCTION Comms_function : result;

CONST Receive  : boolean = FALSE;
      Transmit : boolean = TRUE;

VAR
  I,
  Server_ID,
  Save_Window,
  Status_Window,
  Remote_Window,
  Function_Window   : Byte;
  Verbose : boolean;
  quit : boolean;
  List : EquipmentListType;

  Procedure Update_Status ( Fn, Status : string );

  VAR J : Integer;

  BEGIN
    If Get_Window ( Status_Window ) THEN;
    FOR J := 1 to Comms_Stat_Menu DO BEGIN
      GoToXY ( 18, J );
      ClrEOL;
      WITH Comport [ Current_COM ] DO
      CASE J OF
        1 : Write ( Current_COM );
        2 : Write ( Speed_Msg[ORD( Speed ) + 1 ]);

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

        3 : Write ( Length_Msg[ Length-4] );
        4 : Write ( Parity_Msg[Min(ORD( Parity )+1, 3)]);
        5 : Write ( Stop_Msg[ Stop + 1] );
        6 : Write ( Fn );
        7 : Write ( Status );
      END; { CASE }
    END;
    IF Get_Window ( Function_Window ) THEN;
  END; { Update Status }

  Procedure Reset_remote;
  { This procedure forces the remote server to return to the
    command receive mode}
  BEGIN
    Update_Status ( 'Resetting', 'Please wait. . . ' );
    Xmodm.Send_CAN;
    delay (500);
    Xmodm.Send_CAN;
    delay (500);
    Xmodm.Send_CAN;
    delay (500);
    Xmodm.Send_CAN;
    delay (500);
  END;

  Function Remote_Command ( Command : String128 ) : boolean;

  VAR Ch : char;
      status : result;
      buf : buffer;

    Function stop_case ( status : result ) : boolean;

    BEGIN
      stop_case := ( status = Rx_keypressed )
                OR ( status = Rx_CAN );
               {OR ( status = Rx_done);}
    END;

  BEGIN
    IF Verbose THEN Writeln ('sending command');
    string_to_buf ( Command , buf );


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

    REPEAT
      status := Command_Xfer ( Transmit, buf, 128 );
    UNTIL ( status = Tx_done ) OR ( status = Tx_keypressed );
    CASE status OF
      Tx_CAN,
      Tx_keypressed : BEGIN
                        Update_Status ( '', 'Aborted');
                        IF Get_Window ( Remote_Window ) THEN;
                        WHILE keypressed DO
                          Ch := readkey;
                        Remote_Command := FALSE;
                      END;

      Tx_done       : BEGIN
                        Xmodm.Monitor_transfers := FALSE;
                        IF Verbose THEN Writeln ('Getting response' );
                        REPEAT
                          status := Get_Response ( 128 );
                        UNTIL stop_case ( status);
                        CASE status OF

                        Rx_keypressed :
                          BEGIN
                            Writeln
                             ('Aborted by user waiting for response');
                            delay (1000);
                            WHILE Keypressed DO
                              Ch := readkey; { clear the keypress }
                            Remote_Command := FALSE;
                          END;

                        Rx_done,
                        Rx_CAN        :
                          BEGIN { normally the signal to turn
                                  the link around for the next
                                  command }
                            Remote_Command := TRUE;
                          END;
                        END; { CASE }
                      END;
    END; { CASE }
  END; { Remote Command }


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

  Procedure Rlogin;

  VAR quit : boolean;
      Command : String128;
      buf : buffer;

  BEGIN
    quit := FALSE;
    Update_Status ( 'Login to remote', '' );
    IF Open_Window ( 1, 1, 80, 23, Flag_Borders,
                     'Remote system - ESC terminates') = 0 THEN;
    ClrScr;
    Remote_Window := Active_Window^.ID;
    IF Verbose THEN Writeln ('synchronizing');
    Writeln ('Trying . . .');
    Command := 'Prompt';
    REPEAT
      IF NOT ( Remote_Command ( Command ) ) THEN BEGIN
        Writeln ('Command failed');
        quit := TRUE;
      END
      ELSE BEGIN
        Command := Operator_input ( 'Command ["!<CR>" to quit]',
                                    'Command to send to remote ' );
        IF (Pos ( '!', Command ) <> 0) THEN REPEAT
          Command := Operator_input ( 'Quit', 'Quit? [n, y] ');
          quit := (Command = 'Y') OR (Command = 'y')
               OR ( Command = '' );
        UNTIL (quit OR NOT (Command = 'n') OR NOT (Command = 'N'));
      END;
    UNTIL quit;
    Xmodm.Monitor_transfers := TRUE;
    IF Get_Window ( Remote_Window ) THEN;
    IF Close_Window THEN; { Close the Remote Window }
  END; { Rlogin }

  Procedure Rx_File;

  CONST Curnt_COM : String [ 5 ] = 'COM1';

  VAR Dir : Pathstring;
      Command : String128;
      status : result;
      quit : boolean;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      Ch : Char;
      Settings : byte;
      buf : buffer;

  BEGIN
    quit := false;
    Update_Status ( 'Receive File', '' );

    { Open message window }

    IF Open_Window ( 1, 12, 80, 20, Flag_Borders,
       'Receive File Monitor - press any key to abort') = 0 THEN;
    ClrScr;
    Remote_Window := Active_Window^.ID;
    Command := Operator_input ( 'File to Receive',
                                'Full Path at remote?');
    Writeln ('Trying . . .');
    string_to_buf ( 'zcopy ' + Command + ' ' + Curnt_COM, buf );
    IF Verbose THEN Writeln ('sending command');
    REPEAT
      status := Command_Xfer ( Transmit, buf, 128 );
    UNTIL ( status = Tx_done ) OR ( status = Tx_keypressed );
    WHILE Keypressed DO
      Ch := Readkey;
    IF status <> Tx_done then BEGIN
      Writeln ('Aborted by user on send');
      delay (1000);
      quit := true;
    END
    ELSE  BEGIN
      IF Open_Window ( 1, 1, 80, 25, 0, '') = 0 THEN BEGIN
        ClrScr;
        GetDir ( 0, Dir );
        Exec ( 'zcopy.com', ' ' + Curnt_COM );

      RS_Cleanup;
      RS_Restore ( Current_COM );
        IF Close_Window THEN;
        IF DosError <> 0 THEN BEGIN
          Writeln ('DOS Error ', Error_Code [ DOSERROR ] );
          Delay (2000);
        END;
        System.ChDir ( Dir );
      END;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      IF Verbose THEN Writeln ('Getting response' );
      REPEAT
        status := Get_Response ( 128 );
      UNTIL ( status = Rx_keypressed )
         OR ( status = Rx_CAN );
      IF status = Rx_keypressed THEN BEGIN
        Writeln ('Aborted by user waiting for response');
        delay (1000);
        quit := true;
      END;
      Xmodm.Monitor_transfers := TRUE;
    END;

    { Close message window }

    IF Close_Window THEN;

  END; { Rx_File }

  Procedure Tx_File;

  CONST Curnt_COM : String [ 5 ] = 'COM1';

  VAR Dir : Pathstring;
      quit : boolean;
      Command : String128;
      Ch : Char;
      buf : buffer;
      status : result;

  BEGIN
    quit := false;
    Update_Status ( 'Transmit File', '' );

    { Open message window }

    IF Open_Window ( 1, 12, 80, 20, Flag_Borders,
       'Transmit File Monitor - press any key to abort') = 0 THEN;
    ClrScr;
    Remote_Window := Active_Window^.II;
    Command := Operator_input ( 'File to Transmit',
                                'Full Path (local)? ');
    Writeln ('Trying . . .');
    string_to_buf ( 'zcopy ' + Curnt_COM, buf );

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

    IF Verbose THEN Writeln ('sending command');
    REPEAT
      status := Command_Xfer ( Transmit, buf, 128 );
    UNTIL ( status = Tx_done ) OR ( status = Tx_keypressed );
    WHILE Keypressed DO
      Ch := Readkey;
    IF status <> Tx_done then BEGIN
      Writeln ('Aborted by user on send');
      delay (1000);
      quit := true;
    END
    ELSE  BEGIN
      IF Open_Window ( 1, 1, 80, 25, 0, '') = 0 THEN BEGIN
        ClrScr;
        GetDir ( 0, Dir );
        Exec ( 'zcopy.com ', ' ' + Command + ' ' + Curnt_COM );

      RS_Cleanup;
      RS_Restore ( Current_COM );
        IF Close_Window THEN;
        IF DosError <> 0 THEN BEGIN
          Writeln ('DOS Error ', Error_Code [ DOSERROR ] );
          Delay (2000);
        END;
        System.ChDir ( Dir );
      END;
      IF Verbose THEN Writeln ('Getting response' );
      REPEAT
        status := Get_Response ( 128 );
      UNTIL ( status = Rx_keypressed )
         OR ( status = Rx_CAN );
      IF status = Rx_keypressed THEN BEGIN
        Writeln ('Aborted by user waiting for response');
        delay (1000);
        quit := true;
      END;
      Xmodm.Monitor_transfers := TRUE;
      {IF Close_Window THEN;}
    END;

    { Close message window }

    IF Close_Window THEN;
  END; { Tx_File }

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)


  Procedure Get_Equip;

  VAR Command : string128;
      buf : buffer;

  BEGIN
    Update_Status ( 'Getting remote equipment', '' );
    IF Open_Window ( 1, 1, 80, 23, Flag_Borders,
                     'Remote system - ESC terminates') = 0 THEN;
    ClrScr;
    Remote_Window := Active_Window^.ID;
    IF Verbose THEN Writeln ('synchronizing');
    Writeln ('Trying . . .');
    Command := 'Equip';
    string_to_buf ( Command , buf );
    IF ( Remote_Command ( Command ) ) THEN;
    Xmodm.Monitor_transfers := TRUE;
    IF Close_Window THEN;
  END; { Get_Equip }


BEGIN
  Verbose := TRUE;
  IF Open_Window ( 1, 2, 80, Comms_Stat_Menu + 3, Flag_Borders,
                   'Current Port' ) = 0 THEN;
  Status_Window := Active_Window^.ID;
  ClrScr;
  FOR I := 1 TO Comms_Stat_Menu DO BEGIN
    GoToXY ( 1, I);
    Write ( Comms_Stat [ I ], ':');
  END;
  IF Open_Window ( 41, 2, 75, Comms_Menu + 3, Flag_Borders,
                   'Functions') = 0 THEN BEGIN
    Function_Window := Active_Window^.ID;
    ClrScr;
    Update_Status ( '', '' );
  END
  ELSE Writeln ('Can''t');
  REPEAT
    I := Process_Window_Menu ( Comms_Menu );
    quit := false;
    CASE I OF


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

      0 : ;      { ESC - do nothing }

      1 : BEGIN {Initialize port          }
            Update_Status ( 'Intializing',
                            'Select new parameters' );
            Change_DC_Parameters;
            Save_Window := Active_Window^.ID;
            RS_Cleanup;
            WITH Comport [ Current_COM ] DO
              RS_Initialize ( Current_COM, Speed, Parity,
                              Stop, Length );
            Update_Status ( 'Completed','');
            IF Get_Window ( Save_Window ) THEN;
          END;

      2 : BEGIN {Connect to port          }
            If Open_Window(40,15,47,18,Flag_Borders,'Port') = 0 then;
            ClrScr;
            I:= Process_Window_Menu(Comm_Menu);
            IF I IN [Com1..Com2] THEN BEGIN
              Current_COM := I;
              RS_Cleanup;
              WITH Comport [ Current_COM ] DO
                RS_Initialize ( Current_COM, Speed, Parity, Stop,
                                Length );
              If Close_Window then;
              Update_Status ( 'Connecting', '' );
            END
            ELSE
              Update_Status ( 'Can''t', 'Port out of range' );
          END;

      3 : BEGIN {Disconnect current port }
            Update_Status ( 'Disconnecting', '' );
            RS_Cleanup;
          { Disable those interrupts }
          END;

      4 : BEGIN {Put file to remote       }
            Update_Status ( 'Putting File', '' );
            Tx_File;
          END;


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(*********************  Continue Edwards Excerpt  *********************)

      5 : BEGIN {Get file from remote    }
            Update_Status ( 'Getting File', '' );
            Rx_File;
          END;

      6 : BEGIN {Get machine status      }
            Get_Equip;
          END;

      7 : BEGIN {Login to remote machine }
            Rlogin;
          END;

      8 : BEGIN {Reset remote machine    }
            Reset_remote;
            Update_Status ( 'Reset', '' );
          END;

      9 : BEGIN {(ESC) Exit              }
            I := 0;
          END;
    END; {CASE}

  UNTIL (I = 0) or (quit);
  IF Close_Window THEN;
  IF Close_Window THEN;
  Comms_Function := Tx_done;
END; { Comms_Function }

Procedure Handle_Alt_Key(B:Byte);
{This procedure handles the ALT-Key combinations.

 Input: B - the high order byte returned from Check_Keyboard
}
Var I:Integer;
    S:Long_String;
    status : result;

   Begin
   Case B of
      Alt_A:   Begin   {Drive and path}
               If Open_Window(10,3,50,7,Flag_Borders,'Path') = 0 then;
               ClrScr;

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  *********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                Writeln('Enter new drive and path using format:');
                Writeln('D:\Path\Path...');
                Readln(S);
                If Length(S) > 0 then
                   System.ChDir(S);
                If Close_Window then;
                End;
      Alt_B:    Begin   {Break}
                RS_Break;
                End;
      Alt_C:    Begin    {Clear screen}
                  Modify_Entry(0);
                End;
      Alt_D:    Begin    {Dial}
                Dialing_Directory;
                End;
      Alt_E:    Begin    {Echo}
                Beep(250);
                Echo:=not Echo;
                End;
      Alt_F:    Begin    {Data comm parameters}
                Change_DC_Parameters;
                End;
      Alt_G:    Begin    {Show disk directory}
                  Dirs;
                End;
      Alt_H:    Begin    {Hangup}
                  Beep(250);
                  Hangup;
                End;
      Alt_L:    Begin {DOS Shell}
                  Dos_Shell;
                End;
      Alt_M :   Begin
                  Status := Comms_Function;
                End;
      Alt_P:    Begin
                  Status := Comms_Function;
                End;
      Alt_R,
      PgDn :    Begin    {Receive a file}
                If Ascii_Download then
                   Begin
                   Close(Ascii_File);

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                  Ascii_Download:=False;
                  If Open_Window(35,10,66,13,Flag_Borders,'') = 0 then;
                  ClrScr;
                  Write('Receipt of file terminated');
                  OK('');
                  If Close_Window then;
                  End
               else
                  Transfer_File(False);
               End;
     Alt_S:    Begin    {Activate Server}
                 REPEAT
                   Status := Process_Command;
                 UNTIL ( status = Rx_keypressed ) OR
                       ( status = Tx_keypressed );
               End;
     Alt_T,
     PgUp :    Begin    {Transmit a file}
                 Transfer_File(True);
               End;

     Alt_X:    Begin    {Exit}
               Beep(400);
               End_Emulator:= TRUE;
               If End_Emulator and Ascii_Download then
                  Close(Ascii_File);
               End;
     Home:     Begin    {Help}
               If Open_Window(1,1,29,Min(20,Help_Menu+2),Flag_Borders,
                              'Help')= 0 then;
               ClrScr;
               I:=Process_Window_Menu(Help_Menu);
               If Close_Window then;
               If I > 0 then
                  Handle_Alt_Key(Help_Index[I]);
               End;
     Else      Begin
               Beep(1000);
               End;
     End;  {of Case}
   Build_Status_Line;
   End;  {of Handle_Alt_Key}


(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

Procedure TTY(Ansi:Boolean);
{This procedure provides basic teletype emulation.
 It suppresses NULs and converts GS into a non-destructive backspace}

Const Ansi_Init:Array [0..8] of Char = (#27,'[','3','7',';','4','0',
                                        'm','$');
      Colors:Array [0..7] of Char = ('0','4','2','6','1','5','3','7');
Var Ch:Char;
    I:Integer;
    Regs:Registers;

   Procedure WriteLF;
      Begin
      If not Ansi then
         Writeln
      else if WhereY >= 24 then
         Begin
         Regs.AX:=$0601;
         Regs.CX:=$0000;
         Regs.DX:=$174F;
         Regs.BH:=Background shl 4 + Foreground;
         Intr($10,Dos.Registers(Regs));
         GotoXY(1,24);
         End
      else
         Begin
         Regs.AH:=2;
         Regs.DL:=LF;
         MSDos(Dos.Registers(Regs));
         End;
      End;  {of WriteLF}

   Procedure Writeit(Ch:Char);
      Begin
      If Ansi then
         Begin
         Regs.AH:=2;
         Regs.DL:=Byte(Ch);
         MSDos(Dos.Registers(Regs));
         If WhereY > 24 then WriteLF;
         End
      else
         Write(Ch);
      End;  {of Writeit}

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

   Begin
   If Open_Window(1,1,80,24,0,'') = 0 then; {Save existing screen}
   Build_Status_Line;
   If Ansi then Begin
      Regs.AH:=9;
      Regs.DS:=Seg(Ansi_Init);
      Regs.DX:=Ofs(Ansi_Init);
      Ansi_Init[3]:=Colors[Foreground];
      Ansi_Init[6]:=Colors[Background];
      MSDos(Dos.Registers(Regs));
   End;
   ClrScr;
   Repeat Begin
          Ch:=Check_Auxport;
          Case Byte(Ch) of
             NUL:  ;  {Throw it away}
             GS:   Begin   {Non-destructive backspace}
                   If WhereX > 1 then
                      GotoXY(WhereX-1,WhereY)
                   else if WhereY > 1 then
                      GotoXY(80,WhereY-1)
                   else
                      GotoXY(80,24);
                   End;
             LF:   WriteLF;
             Else  Begin
                   Writeit(Ch);
                   End;
             End;  {of Case}
          I:=Check_Keyboard;
          If I <> 0 then
             If Lo(I) = 0 then
                Handle_Alt_Key(Hi(I))
             else
                Begin
                Ch:=Char(Lo(I));
                RS232_Out(Ch);
                If Echo then
                   Begin
                   Writeit(Ch);
                   If Ch = Char(CR) then
                        WriteLF;
                   If Print then
                      Begin

(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  ********************)

                        Write(LST,Ch);
                        If Ch = Char(CR) then
                           Write(LST,Char(LF));
                        End;
                     End;
                  End;
            End
      Until End_Emulator;
      If Ansi then
         Begin
         Regs.AH:=9;
         Regs.DS:=Seg(Ansi_Init);
         Regs.DX:=Ofs(Ansi_Init);
         Ansi_Init[3]:=Colors[LightGray];
         Ansi_Init[6]:=Colors[Black];
         MSDos(Dos.Registers(Regs));
         End;
      If Close_Window then;
      End;  {of TTY}

{The outer block of Distrib. It performs all necessary initialization
 and presents the user with a list of terminal emulators from which to
 select}

Var I:Integer;
    status : result;
    command_tail : string;

Begin
GetDir(0,Current_Path); (* save current directory for restoration *)
Init_Window_Info;
If Open_Window(1,1,80,25,0,'') = 0 then;
Support.Initialize;
IF ParamCount > 0 THEN BEGIN
  command_tail := ParamStr (1 );
  BumpStrUp ( command_tail );
END;
IF ( ( ParamCount > 0 ) AND ( command_tail = 'SERVER') )
                        OR  ( ParamCount = 0 ) THEN
    REPEAT
      status := Process_command;
    UNTIL  ( status = Tx_keypressed ) OR ( status = Rx_keypressed )



(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
**********************  Continue Edwards Excerpt  ********************)
```

```
(********************  Continue Edwards Excerpt  *********************)

ELSE  BEGIN { Master or maintenance function }
  End_Emulator:=False;
  Emulator:='ANSI';
  TTY(True);
END;
Repeat
until Close_Window;  { Close out all windows }
System.ChDir(Current_Path); (* restore the previous directory *)
End.
```

```
(*  Reprinted with extensive modifications from Advanced Techniques in
    Turbo Pascal by Charles Edwards, by permission of Sybex, Inc.
    Copyright 1987 Sybex, Inc.  All rights reserved.
************************  End Edwards Excerpt  ************************)
```

APPENDIX AB

CONFIGURATION FILE STRUCTURE

A. DISTRIB.CFG FILE STRUCTURE

This is the data structure recorded in the DISTRIB.CFG file when a configuration is saved. This structure can be accessed from the Distrib program main menu by pressing the special key combination Alt-C, for Update Config File.

1. Data Structure for the Default Configuration

This is the data structure in the Support Unit that is recorded in variable Current of type Default_Type.

```
Const Defaults : Default_Type =
     (Default_Name : 'DISTRIB.CFG';   The file name to
modify)
     Default_Com     : 1;    The default communications
port
     Default_Modem   : 2;   The default modem port
     Default_Phone   : '555-1212';
     Default_Speed   : B9600;    The default comm port
speed
     Default_Parity  : None;     The default comm port
parity
     Default_Length  : 8;      The default comm port
     Default_Stop    : 1;      The default comm port
     Default_Echo:False;      Enable Half Duplex
     Default_Textcolor  :  LightGray;    The default text
color
     Default_Menucolor:Green;  The default menu color
     Default_Backcolor:Black;    The default background
color
     Default_Prefix:'ATDT9,,9,,';    The default modem
dialing prefix
     Default_Delay:30);   The default delay to wait for
connection
```

APPENDIX AC

DOCUMENTATION FOR ZCOPY PROGRAM

This is the documentation for the Zcopy program used for file transfer (Flanders, 1989, pp. 251 - 282).

ZCOPY.COM
Command

Bob Flanders
1989 No. 4 (Utilities)

Purpose: Transfers files at high speed, via a serial link, between machines that do not share a common disk format.

Format: ZCOPY source [target] [/w][/n][/u][/o][/a][/p][/d]

Remarks: The two machines must be IBM-compatible and must be connected by a standard "null modem" cable. ZCOPY is executed, with appropriate parameters, on both machines; a 30-second (default) connect timeout is provided.

On the sending machine both a source (filename plus any needed drive and path) and a target (COM1 or COM2) must be specified. ZCOPY supports the * and ? DOS filename "wildcards," but it does not permit renaming files during transfer.
On the receiving machine the source is COM1 or COM2, and the target, if specified, must be a directory path. (Any needed subdirectories must be created on the receiving machine before using ZCOPY.)

The optional /w and /n switches operate before connection is established, and so are entered on the ZCOPY command line of each machine. The /w parameter prolongs the default connection timeout indefinitely; it can be cancelled with Ctrl-Break. The /n parameter sets the highest bit-per-second (bps) rate at which ZCOPY will attempt to transfer data. If used, it must be the same on both machines. The default is /1 (115 kbps). Other acceptable values are /2 through /6 (57.6 kbps, 38.4 kbps, 19.2 kbps, 9600 kbps, and 4800 kbps, respectively). If ZCOPY cannot maintain error-free transfer at a given transfer rate, it automatically steps down to the next lower speed.

The other optional parameters may be entered on either machine's ZCOPY command line. The /u (Update) switch permits overwriting same-named files on the receiving machine without operator confirmation

if the source file is more recent. The /o (Overwrite) switch suppresses the confirmation prompt for all files. By default, when ZCOPY receives a disk-full signal, before aborting it tries to find a smaller selected source file that will fit on the receiving disk. The /a (Abort on Full) aborts at the first disk-full indication. The /p (Pause) switch creates a pause before the transfer operation begins after the connection between machines has been made.

LIST OF REFERENCES

1. Borland International Inc., Turbo Pascal Owner's Handbook Version 4.0, 1987.

2. de Boer, R., <reino@euraiv1.uucp>, info-pascal-@vim.brl.mil message, Subject: Serial Unit in TP4, Message-ID: <797@euraiv1.uucp>, 15 Nov 88 14:17:15 GMT.

3. Defenbaugh, G., "Parents, Children, Redirection, and Piping with DOS Functions 45H and 46H," Programmer's Journal, v. 6, November/December 1986.

4. Duntemann, J., "TURBO Pascal at 4," Turbo Technix, v. 1, November/December 1987.

5. Edwards, C. C., Advanced Techniques in Turbo Pascal, Sybex, Inc., 1987.

6. Flanders, R., "File Transfers Fast and Easy," PC Magazine, v. 8, 28 February 1989.

7. Greco, F.D., "Redirection, or 'They Went That-a-way'", Programmer's Journal, v. 7, January/February, 1987.

8. Greenberg, R.M., "Keeping Up With the Real World: Speedy Serial I/O Processing," Microsoft Journal, v. 2, July 1987.

9. Greenberg, R.M., "TSRCOMM, a Replacement for Interrupt 14", source listing, copyright 1987, Ross M. Greenberg.

10. Hall, W.V., "When Turbo Isn't Enough," in Shammas, N.C., Turbo Pascal Toolbook, M & T Publishing, Inc., 1986.

11. Hartman, R.L., and Yasinsac, A.F., Janus/Ada Implementation of a Star Cluster Network of Personal Computers With Interface to an Ethernet LAN Allowing Access to DDN Resources, M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1986.

12. Kimura, N., <abcscnuk@csuna.uucp>, info-pascal-@vim.brl.mil message, Subject: Re: TP4.0 Aux Problem, Message-ID: <1376@csuna.uucp>, 17 Nov 88 10:20:54 GMT.

13. Krantz, D., "Christensen Protocols in C," Dr. Dobb's Journal, v. 10, June 1985.

14. MacLennan, B.J., Principles of Programming Languages, 2nd. ed., CBS College Publishing, 1987.

15. Mefford, M.J., "Running Programs Painlessly," PC Magazine, v. 7, 16 February, 1988.

16. Microsoft Corporation, MS-DOS Version 3 Programmer's Utility Pack MS-DOS Reference Guide, v. 1, Zenith Data Systems Corporation, 1986.

17. Microsoft Corporation, Microsoft MS-DOS Version 3.21 User's Guide, Zenith Data Systems Corporation, 1987.

18. Norton, P., The Peter Norton Programmer's Guide to the IBM PC, Microsoft Press, 1985

19. Prosise, J., " Instant Access to Directories," PC Magazine, v. 6, 14 April, 1988.

20. RR Software, Inc., JANUS/Ada Package User manuals, 8086 Version 3.2 March 1983, RR Software, 1983

21. Simrin, S., The Waite Group's MS-DOS Bible, 2nd, ed., Howard W. Sams & Company, 1988.

22. Swan, T., Mastering Turbo Pascal Files, Howard W. Sams & Company, 1987.

23. Trimble, R., <reid@hpmtlx.hp.com>, info-pascal@vim.brl.mil message, Subject: Re: xmodem help needed, Message-ID: <5430002@hpmtlx.HP.COM>, 23 Feb 89 21:03:55 GMT.

24. Works, T.V., JANUS/ADA Software Implementation of a Star Cluster Local Area Network of Personal Computers, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1986.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Computer Technology Programs Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor Uno Kodres, Code 52KR Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	9
6.	Mr. Nelson C. Ard 916 Helmsdale Court Chesapeake, VA 23320	4

Thesis
A6455 Ard
c.1 Turbo Pascal implementation of a distributed processing network of MS-DOS microcomputers connected in a master-slave configuration.