APPENDIX A

ASCII CHARACTER SET TABLE

Hex	Dec.	Char.	Hex	Dec.	Char.	Hex	Dec.	Char.	Hex	Dec.	Char.
00	0	NUL	20	32	space	40	64	@	60	96	`
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	'	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DEL

Notes

1. DC1 and DC3 are also known as XON and XOFF, respectively.
2. The **Commodore 64** character set ROMs do not support all characters. The following replacements are made:

$5C (92) \ is replaced by £ (Pound sterling currency symbol)
$5F (95) _ is replaced by ← (Left pointing arrow)
$60 (96) ` is replaced by − (Horizontal bar - not the minus sign)
$7B (123) { is replaced by + (Cross - not the plus sign)
$7C (124) | is replaced by ◧ (left half checkerboard)
$7D (125) } is replaced by | (vertical bar)
$7E (126) ~ is replaced by ▒ (checkerboard)

This page is intentionally left blank

APPENDIX B

PROMAL KEY CODES RETURNED BY FUNCTIONS GETKEY & TESTKEY (HEX)

Table B-1: Commodore 64

Key Legend	Plain	Shift	CTRL	C=	Remarks
<-- (left arrow)	5F	5F	06	5F	(next to 1 key)
1 / !	31	21	90	81	BLK
2 / "	32	22	05	95	WHT
3 / #	33	23	1C	96	RED
4 / $	34	24	9F	97	CYN
5 / %	35	25	9C	98	PUR
6 / &	36	26	1E	99	GRN
7 / '	37	27	1F	9A	BLU
8 / (	38	28	9E	9B	YEL
9 /)	39	29	12	29	RVS ON
0	30	30	92	30	RVS OFF
+	2B	DB	--	A6	
-	2D	DD	--	DC	
British currency £	5C	A9	1C	A8	(pounds ster.)
CLR HOME	13	93	--	93	
INST DEL	14	94	--	94	
Q	71	51	11	AB	
W	77	57	17	B3	
E	65	45	05	B1	
R	72	52	12	B2	
T	74	54	14	A3	
Y	79	59	19	B7	
U	75	55	15	B8	
I	69	49	09	A2	
O	6F	4F	0F	B9	
P	70	50	10	AF	
@	40	BA	--	A4	
*	2A	C0	--	DF	
up arrow	5E	DE	1E	DE	
RESTORE	--	--	--	--	
A	61	41	01	B0	
S	73	53	13	AE	
D	64	44	04	AC	
F	66	46	06	BB	
G	67	47	07	A5	

Table B-1 continued: Commodore 64

Key Legend	Plain	Shift	CTRL	C=	Remarks
H	68	48	08	B4	
J	6A	4A	0A	B5	
K	6B	4B	0B	A1	
L	6C	4C	0C	B6	
: / [	3A	5B	1B	5B	
; /]	3B	5D	1D	5D	
=	3D	3D	1F	3D	
RETURN	0D	8D	--	8D	
Z	7A	5A	1A	AD	
X	78	58	18	BD	
C	63	43	03	BC	
V	76	56	16	BE	
B	62	42	02	BF	
N	6E	4E	0E	AA	
M	6D	4D	0D	A7	
, / <	2C	3C	--	3C	
. / >	2E	3E	--	3E	
/ / ?	2F	3F	--	3F	
CRSR up down	11	91	--	91	
CRSR <= =>	1D	9D	--	9D	
f1 / f2	85	89	--	89	
f3 / f4	86	8A	--	8A	
f5 / f6	87	8B	--	8B	
f7 / f8	88	8C	--	8C	
space	20	20	--	20	
RUN STOP	03	83	*	83	

Notes:

CTRL/RUN STOP aborts program to the PROMAL EXECUTIVE.
SHIFT/C= switches mode (upper & lower <--> upper & graphics).
Codes shown assume upper & lower case mode.
Alpha-lock (CTRL-A) affects GETKEY but not TESTKEY codes.

Table B-2: Apple II

Key Legend	Plain	Shift	CTRL	Apple	Shift/Apple
ESC	1B	1B	1B	9B	9B
1 / !	31	21	31	B1	A1
2 / @	32	40	32	B2	C0
3 / #	33	23	33	B3	A3
4 / $	34	24	34	B4	A4
5 / %	35	25	35	B5	A5
6 / ^	36	5E	1E	B6	DE
7 / &	37	26	37	B7	A6
8 / *	38	2A	38	B8	AA
9 / (	39	28	39	B9	A8
0 /)	30	29	30	B0	A9
- / _	2D	5F	2D	AD	DF
= / +	3D	2B	3D	BD	AB
DELETE	7F	7F	7F	FF	FF
TAB	09	09	09	89	89
Q	71	51	11	F1	D1
W	77	57	17	F7	D7
E	65	45	05	E5	C5
R	72	52	12	F2	D2
T	74	54	14	F4	D4
Y	79	59	19	F9	D9
U	75	55	15	F5	D5
I	69	49	09	E9	C9
O	6F	4F	0F	EF	CF
P	70	50	10	F0	D0
A	61	41	01	E1	C1
S	73	53	13	F3	D3
D	64	44	04	E4	C4
F	66	46	06	E6	C6
G	67	47	07	E7	C7

Table B-2 Continued - Apple II

Key Legend	Plain	Shift	CTRL	Apple	Shift/Apple
H	68	48	08	E8	C8
J	6A	4A	0A	EA	CA
K	6B	4B	0B	EB	CB
L	6C	4C	0C	EC	CC
; / :	3B	3A	3B	BB	BA
' / "	27	22	27	A7	A2
RETURN	0D	0D	0D	8D	8D
Z	7A	5A	1A	FA	DA
X	78	58	18	F8	D8
C	63	43	03	E3	C3
V	76	56	16	F6	D6
B	62	42	02	E2	C2
N	6E	4E	0E	EE	CE
M	6D	4D	0D	ED	CD
, / <	2C	3C	2C	AC	BC
. / >	2E	3E	2E	AE	BE
/ / ?	2F	3F	2F	AF	BF
` / ~	60	7E	60	E0	FE
<--	08	08	08	88	88
-->	15	15	15	95	95
down arrow	0A	0A	0A	8A	8A
up arrow	0B	0B	0B	8B	8B

NOTES:
Alpha lock (CTRL-A) affects GETKEY but not TESTKEY.

APPENDIX C

ERROR MESSAGES AND MEANINGS

Arranged alphabetically (including punctuation)

*** ERROR: ALREADY LOADED

You have attempted to GET a program which is already in memory. If this is deliberate, UNLOAD the program first and then re-issue the GET command.

*** ERROR: DEVICE NOT READY

Usually this indicates that the disk drive door is not closed. Also check for a disconnected or off-line printer or disk drive, unformatted disk, etc. On the Apple II, this error probably indicates that you changed diskettes without issuing a PREFIX command to select the new volume (PREFIX * can fix this).

*** ERROR: DISK ERROR

This uncommon error message indicates a hardware read or write error on the disk. Check for a disk drive turned off or not connected, etc. It may indicate that your diskette has become damaged or that the disk drive heads need to be cleaned. It may also indicate that part of the operating system has been wiped out in memory by an errant program, or similar difficulties. It may also indicate an attempt to append a write-protected disk or locked file.

*** ERROR: FILE DOES NOT EXIST

You have issued a command to the EXECUTIVE to act on a file which does not exist on the currently selected disk. Remember that the default file extension is ".C". If you are trying to act on a file which does not have a file extension remember to enclose the name in quotes (case sensitive on the Commodore 64). For the Commodore 64 COPY command, if the file is not type SEQ, the type must be specified after a comma inside the quotes, for example "BASICPROG,P". On the Apple II, you may have switched disks without a PREFIX * command.

*** ERROR: FILE ALREADY EXISTS

You have issued an EXECUTIVE command which tried to write a new file with the same name as an existing file. If this was deliberate, DELETE the existing file and try again. Remember that the default file extension is .C. For file names without extensions, the name should be enclosed in quotes.

*** ERROR: ILLEGAL COMMAND SYNTAX

You have issued an EXECUTIVE command with arguments which do not conform to the requirements. Consult the PROMAL USER´S GUIDE for the proper command syntax.

*** ERROR: ILLEGAL FILE/DEVICE NAME
You have issued a command with an illegal file or device name. Check the file naming conventions described in the first part of the PROMAL USER´S MANUAL. For non-conforming files, you must enclose the file names in quotes when using the EXECUTIVE. For the Apple, make sure you are not trying to copy between two disks with the same volume name.

*** ERROR: ILLEGAL OPEN DIRECTION
You have issued a command which tries to output to an input-only device, or visa-versa.

*** ERROR: NOT ENOUGH FREE DISK
You have issued an EXECUTIVE command which has tried to write a file to disk larger than the remaining disk space, or, there are no more free buffers (Apple II) or channels (Commodore 64) available.

*** ERROR: NOT IMP.
The command you have issued is not implemented on your version of PROMAL.

*** ERROR: WRITE PROTECTED
You have issued a command which attempted to write or alter a write-protected disk (or a locked file on the Apple). If you wish, remove the write protect sticker or UNLOCK the file and try again.

*** RUNTIME ERROR: 0 DIVIDE
Division (or %) by zero, or an arithmetic overflow has occurred. If this error occurred during compilation, then a REAL literal number was specified which was out of range. The largest REAL number is approximately 1E+37.

*** RUNTIME ERROR: I-O ILLEGAL DIRECTION
A library routine was called to input or output to a file or device which was opened in a different mode; for example, trying to input from the printer.

*** RUNTIME ERROR: ILLEGAL / UNOPEN FILE HANDLE
A library routine expected to find an open file handle for the first argument, but did not find one. Check for a missing file handle where required. Check to make sure you have properly opened the file or device and have saved the file handle in a WORD type variable. Make sure you have not already closed the handle. An OPEN function call should always be tested for success. If the function returns 0, it was not successful.

*** RUNTIME ERROR: ILLEGAL # ARGS - LIB. CALL
A library PROC or FUNC or machine language routine was called with an invalid number of arguments. Check the LIBRARY MANUAL for the correct arguments required.

*** RUNTIME ERROR: ILLEGAL ARG, LIB. CALL
A library routine was called with an invalid or out-of-range argument. Make sure you are using the appropriate type arguments (e.g., not using a REAL where a WORD is expected). Check the LIBRARY MANUAL for restrictions on arguments.

*** RUNTIME ERROR: ILLEGAL I-O REDIRECTION
An illegal I-O redirection has been made. Only STDIN, STDOUT, and STDJOB may be redirected, and they must be redirected to an open file or device. Check for a missing # in front of STDIN or STDOUT. See REDIRECT in the LIBRARY MANUAL for further information.

*** RUNTIME ERROR: ILLEGAL OPCODE
A program has attempted to execute a non-existent instruction. Make sure you are not trying to use REAL arithmetic after a NOREAL command. It can also be caused by a program destroying itself by writing data into its code space. Check for bad pointers, arrays out of bounds, using a value where an address is required, etc. If this error occurs during compilation, you have attempted to compile a program using REAL data after executing a NOREAL command.

*** RUNTIME ERROR: M/L BREAK
A machine language BRK ($00) instruction has been executed. If this is not expected, it is often a symptom of a piece of a program (probably the PROMAL library) having been zeroed by an errant program. It may also reflect an erroneous definition of an EXTernal routine or failure to load a required software package.

*** RUNTIME ERROR: PROMAL BREAK
A PROMAL PROGRAM (possibly the EDITor or EXECUTIVE) has encountered a $00 instruction at the indicated address. This usually indicates that an program bug has caused part of the program to become zeroed out. Check for bad pointers, arrays out of bounds, using a value where an address is required, etc.

*** RUNTIME ERROR: REQ'D PROGRAM NOT LOADED
A required software package is not in memory. Check to see if you are trying to use REAL arithmetic after a NOREAL command. It may also reflect a defective EXTernal declaration. If this error occurs during compilation, you are trying to compile REAL data after a NOREAL command.

*** RUNTIME ERROR: STACK ERROR
The stack has overflowed. This may indicate that you have a routine which calls itself indefinitely or a recursion error. It may also indicate subroutines nested too deeply or with too many arguments passed. If this error occurred during compilation, you have a statement with an expression which is too complex to compile due to stack limitations (for example, 12 levels of parentheses). This may be aggravated by having many levels of indentation, and by using function calls in the expression. In this case, use intermediate temporary variables for sub-expressions to reduce the complexity of the statement.

ERROR 1:
Illegal character here

The compiler has encountered a character which cannot legally be present at this point in the statement. Check for a missing or extra punctuation mark. If the source file was created by something other than the PROMAL EDITor, check for an embedded tab, linefeed, or other invisible control character (DUMPFILE may help find it).

ERROR 2:
Illegal character constant

A character constant must be a single character enclosed by single quotes (´). Check for missing quote or more than one character. The quote character itself can be written as ´´´´.

ERROR 3:
Illegal string constant

A string constant must be enclosed on both ends by double quotes ("). It may not cross a line boundary. Check for unbalanced quotes. The double quote character can be written inside a string as "".

ERROR 4:
PROGRAM or OVERLAY expected

Your program must start with a PROGRAM statement (or OVERLAY statement). Make sure you are compiling the right file.

ERROR 5:
<Name> expected

The compiler expected to find an identifier (name) at this point in the statement. Make sure you are not trying to use a reserved word as a name.

ERROR 6:
Duplicate name

The identifier has already been declared previously. Make sure you are not trying to define a name that is already defined in the LIBRARY, INCLUDE file, etc. Also make sure that you don´t have a variable that duplicates a procedure, function or data name, or visa-versa. In some circumstances, this error may refer to an identifier on the line above the one shown.

ERROR 7:
= expected

The compiler expected to see an "=" operator at this point in the statement. Make sure that you are not trying to use a simple (un-subscripted) variable as an array, or as a procedure name.

ERROR 8:
Constant expected

The compiler expected to see a constant at this point in the statement. Be sure you are not trying to use a variable name where a constant is required. Remember that you may not have constants of type REAL. Also check to make sure that the value specified for the constant is not out of range (Note: the offending constant may be slightly beyond the row of asterisks on the compiler error message). For example, $100000 is an out of range constant.

ERROR 9:
] expected

The compiler expected to find a right bracket at this point in the statement. Remember that square brackets, not parentheses, are used to delimit PROMAL array subscripts. Check for a missing comma. Remember that you should not specify the size of a DATA or EXTernal array.

ERROR 10:
Illegal data type

The expression does not meet the required data type. Remember that in a DATA declaration defining a string, the type is WORD, not BYTE, because the result is a pointer to the string. Also remember that a FOR-statement index must be a simple variable of type WORD. The choices on a CHOOSE statement must match the type of the expression following the CHOOSE **exactly** (you may need a type cast to make a small numeric constant match a word or integer variable, for example 1:+), and may not be type REAL. Finally, keep in mind that the boolean operators AND, OR, NOT, and XOR operate only on type BYTE.

ERROR 11:
Illegal subscript

Subscripted variables may not be used for local variables or arguments.

ERROR 12:
Variable name expected

The compiler expected to find a variable name at this point in the statement. Make sure that you are not trying to use a reserved word, procedure, function name, or constant for a variable. DATA names may not be the destination for an assignment statement.

ERROR 13:
) expected

The compiler expected a right parentheses at this point in the statement. Check for too many or too few arguments on a function call, or missing or unbalanced parentheses. Also make sure you are not trying to enclose the argument list for a procedure call in parentheses.

ERROR 14:
Illegal expression

The expression does not follow the syntax diagram in Appendix P of the PROMAL LANGUAGE MANUAL. Check for an illegal sequence of operators, missing punctuation, etc. Note that the indirect operators (@<, @+, @-, @.) may not appear after the variable name for an assignment statement (use the global array M instead).

ERROR 15:
# is illegal here

The # address operator cannot be used here. The address operator cannot appear on the left hand side of an assignment statement, nor can it be applied to anything except a variable. The # operator must directly precede the variable name.

ERROR 16:
Type name expected

The compiler expected to see BYTE, INT, WORD, or REAL at this point in the statement. The type indicator must precede the variable or function name.

ERROR 17:
BEGIN expected

The statement is illegal at this point in the program. If you have a declaration, check the first word for spelling. If this is an executable statement, you must have a BEGIN statement first.

ERROR 18:
End of line expected

This is a general error message indicating that the compiler could not construct a legal statement with what you have at this point in the line. Check for: too many arguments on a procedure call, subscripts on a simple variable, etc. Also be sure you didn´t forget the ";" which must precede a comment.

ERROR 19:
, expected

The compiler expected a comma at this point in the statement. Check for too few subscripts on an array reference, or too few arguments on a function or procedure call.

ERROR 20:
Illegal type name here

The type (BYTE, INT, WORD, or REAL) indicated is inconsistent with prior usage.

ERROR 21:
Not in WHILE or REPEAT loop

The BREAK or NEXT statements may only be used inside a WHILE or REPEAT loop.

ERROR 22:
Statement expected

This is a general error message indicating that the compiler was expecting an executable statement but did not find one. Check for a missing END statement in a prior procedure or function or a misplaced declaration.

ERROR 23:
Wrong # of arguments

The procedure or function call has too many or too few arguments.

ERROR 24:
Indentation error

The statement starts with the wrong indentation. Each level of indentation must be **exactly** two blanks. A statement following a conditional statement must be indented. If this is a statement terminating a conditional block such as an ELSE or UNTIL, it should not be indented as far as the line immediately above it. Each choice of a CHOOSE statement should be at the same level of indentation as the original CHOOSE keyword; the statements executed for each choice should be indented one level. Check also for a missing ELSE for a choose statement, which is always required.

ERROR 25:
UNTIL expected

A preceding REPEAT statement is not balanced by an UNTIL at the same level of indentation.

ERROR 26:
Unexpected end of file

The compiler reached end-of-file without having reached the END statement in the main program. Check for a missing END statement or INCLUDE statement.

ERROR 27:
Undefined

The identifier indicated has not been previously declared or defined. All variables, constants, procedures and functions must be declared before they are referenced. Check for a missing INCLUDE LIBRARY or other INCLUDE statement, or for a spelling error.

ERROR 28:
Illegal FOR variable

The index variable for a FOR-loop must be a simple (non-subscripted) variable of type WORD.

ERROR 29:
TO expected

The compiler expected the keyword TO to appear at this point in the FOR statement.

ERROR 30:
[expected

The compiler expected to find a left bracket at this point. Remember that square brackets, not parentheses, are used to delimit PROMAL arrays. Make sure you are not trying to use an array name on the left side of an assignment statement without specifying which element of the array should get the result.

ERROR 31:
PROC or FUNC expected

The compiler expected to see the keyword PROC or FUNC at this point in the declaration. This error can also be caused by a missing type indicator (BYTE, INT, or WORD) on an EXTernal variable declaration.

ERROR 32:
AT expected

The COMPILER expected to find the keyword AT at this point in the declaration.

ERROR 33:
Illegal refuge

The keyword REFUGE must be followed by a constant of value 0, 1, or 2.

ERROR 34:
Illegal REAL constant

A literal number is incorrectly formed. Check for the letter O or l instead of zero or one, missing ´.´, etc. On the Commodore 64, check for cross or bar characters instead of + or -.

ERROR 35:
Non-REAL expected

An expression of type REAL is not legal at this point. Subscripts must be type WORD. CHOOSE statements may not have an expression of type REAL. CON declarations may not be REAL (use DATA instead).

ERROR 36:
Illegal import

You have more IMPORT files than are allowed (maximum is 6), or have an erroneous declaration in an imported block.

ERROR 37:
Illegal <import var> :> = ...

You may not use the high-byte operator (:>) on an imported variable appearing on the left hand side of an assignment statement.

ERROR 38:
Illegal export

The declaration cannot be EXPORTed. Only constants, variables, data, procedures and functions can be EXPORTed. You may not EXPORT EXTernal declarations. Check for missing EXPORT keyword on PROGRAM line.

ERROR 39:
Too many dimensions

PROMAL arrays may have a maximum of eight dimensions.

ERROR 40:
Demo compiler can´t IMPORT/EXPORT

The compiler on the PROMAL DEMO diskette does not support EXPORT declarations or INCLUDE files of IMPORTs. You must use the full compiler for these features.

ERROR 129:
Compilation cancelled

This is not an error message, but indicates that compilation was terminated by the operator in response to a prompt.

ERROR 130:
Not enough free memory

The COMPILER cannot find enough free memory for its tables. UNLOAD some programs and try again. On the Commodore-64 using the standard compiler, you will not be able to compile unless the workspace is clear (but you can with the demo compiler). On the Apple II, if you have issued a BUFFERS HIRES, you will need to give a BUFFERS 3 command before compiling.

ERROR 131:
Cannot open object file
The compiler cannot open the object file for writing. Check for a write-protected diskette or full diskette. On the Apple, check for a locked file or diskette change without PREFIX command.

ERROR 132:
Cannot open source file
The compiler could not find the specified source file, or could not successfully open it for reading (for example, drive not ready). Check for spelling errors. The default extension for source files is ".S". On the Apple, check for disk changed without PREFIX * command.

ERROR 133:
Cannot open list file
The compiler could not open the listing file for writing. Check for device not ready, write-protected disk, disk full, etc. The default extension for the list file is ".L".

ERROR 134:
Cannot open export file
The compiler could not open the export file for writing. Make sure that the disk is not write-protected or full (or the file locked on the Apple). The Commodore 64 may not be able to open the export file if you have a listing enabled (due to limitations of the 1541 drive).

ERROR 135:
Cannot open include file
The compiler cannot find or cannot open the specified INCLUDE file for reading. Make sure the desired file is present on the disk. The default file extension is ".S". On the Apple II, this may be caused by not having the proper prefix (volume name). Also, you may have to increase the number of buffers on the Apple if you have a listing file, export file, and/or nested INCLUDE files.

ERROR 136:
No source file, Workspace empty
No source file name was specified on the COMPILE command, and the Workspace is empty. You need to specify a filename to be compiled.

ERROR 137:
Illegal COMPILE argument
The COMPILE command has an illegal argument. See the PROMAL USER´S GUIDE for the correct command syntax.

ERROR 138:
File name duplicates another argument
One of the output file names specified on the COMMAND line is the same as one of the input files, including the extension.

ERROR 139:
Can´t write to L device
The L device was specified as an output file for the compiler. This is not permitted.

ERROR 140:
Can´t write to Workspace

The W device was specified as an output file for the compiler. This is not permitted.

ERROR 141:
String buffer overflow (Use B option)

The string buffer (literal pool) used by the compiler has overflowed. UNLOAD all programs and use the B option on the COMPILE command. If you have already done this, use the B=Solf option to increase the size of the literal pool (See USER´S GUIDE).

ERROR 142:
Forward Reference overflow

The forward reference table used by the compiler has overflowed. UNLOAD all programs and use the B option on the COMPILE command. If you have already done this, use the B=Solf option to increase the size of the forward reference table.

ERROR 143:
Object buffer overflow (Use B option)

The object buffer used internally by the compiler has overflowed. UNLOAD all programs and use the B option on the COMPILE command.

ERROR 144:
Symbol table overflow (Use B option)

The internal symbol table used by the compiler has overflowed. UNLOAD all programs and use the B option on the COMPILE command. if you have already done this, use the B=Solf option to increase the size of the symbol table.

ERROR 145:
Too many ELSEs

Your program has an IF with more ELSEs than the compiler can handle, or nested loops greater than it can handle.

ERROR 146:
Too many nested loops

Your program has loops nested to a greater depth or complexity than the compiler can handle.

ERROR 147:
INCLUDEs overnested

You have an INCLUDE statement inside an INCLUDE file, which requires opening more files than the Commodore 64 disk or Apple ProDOS will allow.

ERROR 148:
Unbalanced ? (conditional comp.)

You have a ? in column 1 of a statement initiating a conditional block which is not balanced by a matching ? terminating the conditional compilation block. Conditional compilation blocks may not be nested.

ERROR 149:
1st sector rewrite error

For the Commodore 64, this indicates a hardware or firmware disk drive failure or incompatibility. Don´t use 2-sided mode on a 1571.

ERROR 150:
B option not in Demo compiler

The B compiler option is not supported in by the PROMAL DEMO COMPILER. You must use the full compiler (on the System Disk) to use the B option.

ERROR 151:
Demo compiler line limit exceeded

The Demo compiler can only compile files with up to 400 lines, excluding comments (but including the LIBRARY). You need to use the full compiler on the PROMAL system disk.

ERROR 152:
Disk write error

The compiler encountered a disk error while writing a file.

ERROR 153:
Disk full

There was not enough room to write the file on the disk.

xxxx ISN´T A TEXT FILE

You have tried to EDIT a file named xxxx which the EDITor thinks is not a text file. Check to see if you are trying to edit a compiled program or data. It may also indicate that the file has lines longer than 125 characters. For the Apple II, it may indicate that the file was prepared with a word processor which sets bit 7 of each character to 1. If this is the case, you can fix the file by using the CLEARBIT7 demo program.

NO BUFFER SPACE

The EDITor could not find enough free space for its buffer. To correct this, type UNLOAD (and WS 0 if you are not using the Workspace on the Commodore 64), and try again.

NOT A PROMAL OBJECT FILE: xxxx

You have attempted to execute a file which is not a compiled PROMAL program or a relocatable machine language program. All PROMAL programs must be successfully compiled before they can be executed. This error usually occurs when you try to execute a program which was compiled using the B option but had compilation errors. It can also occur if you attempt to execute a version 2.0 module on a version 1.X PROMAL system.

NOT ENOUGH FREE MEMORY

The specified program or overlay could not be loaded because there is not enough free memory. If it is a program on the Commodore 64, you will need to set the Workspace size to 0 and try again. If it is an overlay, you need to unload all programs and restart the main program. If the problem persists, your program may simply be too large. Consider modifying it to use overlays, or, use the NOREAL command if appropriate, to free up more memory. Remember that your variables also require memory; you should consider reducing the size of your arrays. For the Commodore 64, you can gain a lot of space by using a bootstrap loader to set HIFREE and HIMEM to MEMLIM and then loading your program as described in the section on the LOADer.

NOT LOADED OR RELOC ERROR: xxxx

You have attempted to load or execute a program or overlay which imports from the indicated program or overlay, without having that program or overlay loaded first. UNLOAD memory and use the GET command to load any programs needed. You may wish to write a bootstrap program as described in the LOADer section of the PROMAL LANGUAGE MANUAL to load the required modules automatically. If the program name shown is the same as the program you are trying to execute, then there is not enough free memory to relocate your program after loading it. You need to unload other programs or free up additional memory as described above.

OK TO CLEAR WORKSPACE (Y/N)?

This Commodore 64 message is not an error message but a warning. It is given if you specified the B option on the compiler, but there is something in the Workspace. If you reply with a Y, the compiler will clear the workspace and proceed. Otherwise, it will abort.

PROGRAM OR OVERLAY NOT FOUND: xxxx

You have issued an EXECUTIVE command which is not a built in command, nor is it in memory or on disk. Check for spelling errors, and make sure you have the correct diskette. Remember that ".C" will be the default file extension. For the Apple II, you may have changed disks without using the PREFIX command to set the new volume name.

xxxx TOO LARGE TO EDIT

The specified file name, xxxx, is too large to EDIT in the available memory space. To correct this, type UNLOAD to free additional memory and try again. If you are not using the Workspace on the Commodore 64, you should also issue a WS 0 command. If you have already done all this, your file may be too large to EDIT. You can split it into two files using the SPLIT Utility, and then edit each file separately.

USER BREAK

This is not necessarily an indication of an error, but shows that the program was aborted by the user (by CTRL-STOP on the Commodore 64 or by CTRL-C or CTRL-RESET on the Apple).

APPENDIX D

LOCATING RUNTIME ERRORS AND VARIABLES IN MEMORY

The PROMAL nucleus provides runtime checking for many errors such as division by zero, illegal arguments on Library routines, etc. A typical runtime error message would be:

```
*** RUNTIME ERROR: ILLEGAL # ARGS - LIB CALL
AT $72B2
```

This tells you that you attempted to call a Library routine with too many or too few arguments. But where in your program did this occur? The absolute address is given in the error message as $72B2. To find the offending statement in your listing, proceed as follows:

1. Execute a MAP command from the EXECUTIVE.

2. Locate your program's starting address (e.g. "AT 7100") in the MAP display.

3. Subtract this value (using hex arithmetic) from the address displayed with the error message. The result is the relative address from the start of your program, for example $72B2 - $7100 = $01B2.

4. Using your program listing, find the statement (not a variable or data declaration) with an address (shown in the column to the left of the statement) that spans the calculated address. This statement (or possibly, the preceding or next statement) is the one where your error occurred.

You can use a similar technique to DUMP the value of shared global variables and global scalar variables (but not local variables). Shared global variables are arrays of any type, or REAL variables (both simple and arrays). Global scalar variables are non-array BYTE, INT and WORD variables which are not declared inside a PROC or FUNC.

To locate a global scalar variable, add the address shown to the left of the variable's declaration on the listing to the address shown as the starting address for variables in your map display. For example, if your listing shows:

```
8   BYTE MYVAL
```

and the MAP for your loaded program shows:

```
MYPROG     (PRO.)  9/ 3/85 CHKSUM 4B9D
 AT 7100-73FF (VARS: A100-A2FF)
```

then add $08 to $A100, giving $A108. This is the absolute address of your variable, MYAL = $A108. Note that if the variables start at an address above the "SYSTEM SPACE" location on the Apple II, you will not be able to DUMP the correct value of your variable because the EXECUTIVE has re-used that address space.

Locating an array or REAL variable is slightly more complex. Use the SIZE command (or the summary at the end of the listing) to determine the number of bytes of scalar variables used by your program. For example, if the SIZE command displays:

```
   MYPROG      (PRO.) 9/ 3/85 VER.2
    CODE $20BA, GLOB VARS $01C0, $09
```

then your program has $09 bytes of scalar variables. Add this number to the address shown in the listing for your variable, and add that result to the starting address of variables shown for the map command. For instance, using the above example, if your listing shows:

```
   6A  WORD VALUES [5]
```

then the absolute address of VALUE[0] is at $A173 ($6A + $09 +$A100).

When displaying the value of variables, remember that WORD and INT variables are stored with the low order byte first and the high order byte at the next higher address.

REAL variables occupy 6 bytes each, in the following format:

```
        |S<--Exp--><---------------Mantissa----------------->|
Addr:      +0        +1        +2        +3        +4        +5
```

This format is based on the IEEE standard single precision data format, but is extended with 2 additional bytes in the low order mantissa to increase the accuracy from 6 to 11 significant decimal digits. It uses a binary representation where:

S	=	1 bit sign (1=negative value)
Exp	=	8 bit exponent (biased by $7F, 0 if number=0.0)
Mantissa	=	39 bit mantissa normalized between 1.0 and 2.0, with an implied 1 bit & binary point to left of mantissa.

If you are not familiar with floating point representations, you may wish to consult an "elementary" book on computer arithmetic, or the IEEE Floating Point Standard before attempting to interpret the value.

Note for numerical analysts: The PROMAL floating point routines do not support gradual underflow. Any number with an exponent of zero is considered zero. Also, rounding-to-even is not supported (but rounding is). This is of no consequence to normal users.

APPENDIX E

PRINTER SUPPORT

COMMODORE 64 PRINTER SUPPORT

PROMAL supports standard Commodore Printers using the serial interface. All Commodore and Commodore-compatible printers we tested worked without any special effort as the "P" device with PROMAL. Parallel printers using Cardco (Model C through G+) or similar adapters should also function normally.

Printers (or printer interfaces) often have special modes selected on the basis of the "secondary address". The following three variables can be used to control your printer:

```
EXT ASM BYTE C64PSA AT $0DF3  ; Desired secondary address (default 7)
EXT ASM BYTE C64PUL AT $0DF4  ; Bit 7=1=flip case (default=$80=yes)
EXT ASM BYTE C64PDV AT $0DF5  ; C-64 printer device # (default 4)
```

You should set these variables to the desired choices before OPENing the P device. The C64PUL variable controls whether or not lower case and upper case alphabetic characters should be reversed before output to the printer. This is normally needed because PROMAL uses standard ASCII characters but most Commodore-compatible printers expect "Commodore ASCII". If your printer prints alphabetic characters in the wrong case, you can use a **SET DF4 0** command from the EXECUTIVE, or in your BOOTSCRIPT.J file.

The standard Commodore device number for the printer is 4. However, if you have a second printer on the serial bus or are using a plotter, you may wish to open the P device to a different device number. You can do this by installing the desired device number in the byte at $0DF5 (C64PDV).

The Commodore 1525 printer does not support form feeds, so listings will not be properly paginated, but 1526 printers will work properly.

It has been reported that some versions of the Commodore 1526 printer have intermittent problems when used with the 1541 disk drive. These problems are characterized by a serial bus "lockup", which may cause the system to hang inexplicably or to display error number 40 or 41. **This problem has nothing to do with PROMAL** and will also appear with other software. Rather, this is a problem with the Commodore ROMs in the printer and/or disk drive. If you experience these problems, you may want to contact Commodore dealer and request that he upgrade your system to the latest level ROMs. If in doubt, you can find out what ROMs you have in your printer by performing the 1526 self-test. If the prints "CBM COMMODORE 1526/MPM-802 PRINTER - REV 07C", then you have the latest printer ROMs. At the time of publication of this manual, our best information is that the latest ROMs are as follows:

1541 Disk Drive ROM part number 901229-05
1526 Printer ROM part number 325341-08

(Our sincere thanks to Mr. A. Ryan of Ontario who provided this information).

APPLE II PRINTER SUPPORT

PROMAL supports standard Apple printers or compatible printers. For the IIe, the printer card should be installed in slot 1 and conform to the standards for Apple Pascal. For the Apple IIc, the printer should be attached to the printer port (port 1) in the usual fashion.

You can control whether or not PROMAL should automatically send a LF after every CR to your printer by the setting of the following variable:

```
EXT BYTE APLPALF AT $0DF3  ; Bit 7=1=send LF after CR
```

If your printer double spaces when it should single space, set this variable to 0. This can be done from the EXECUTIVE or a JOB file with a **SET DF3 0** command. If it prints one line on top of the other, set it to $80. When outputting graphics or specuial escape sequences, you may need to turn this off (so a $0D graphic data byte won't be interpreted as a CR and cause a suppious $0A linefeed data byte to be sent to the printer).

Also, if your computer is a IIc or is connected by a **serial** interface, you will need to set another variable to 0 to perform graphics or escape sequences. This is not a PROMAL variable, but a global Apple variable that controls the ROM output routines in the Apple:

```
EXT BYTE PRESCCH AT $0638+$C0+1 ; Serial command enable flag, Apple
...
PRESCCH = 0 ; Disable Apple ROM processing of serial printer output
```

This will keep the Apple from processing escape sequences to the serial port internally, and will pass them straight through to the printer.

PROMAL automatically configures an appropriate printer driver for your computer during boot-up. In very rare cases, if you are using a printer card which does not follow the Apple standard, you may have to supply your own printer driver. In this case, See **APPENDIX G**, which tells the location of a pointer to a table of addresses for the printer driver input and output vectors. The table pointed to consists of a WORD holding the address of the initialization entry point, followed by a WORD holding the addres of the output entry point (will be called with character in A).

APPENDIX F

DATA COMMUNICATIONS SUPPORT

PROMAL provides support for serial data communications using RS-232C asynchronous data transmission by the T device. Input and output for EXECUTIVE commands can be redirected to the modem in the same way as to the printer or other device. More frequently, a PROMAL program will perform input and output to the T device. This makes it relatively easy to handle roughly 90 percent of your telecommunications needs. This section assumes you have a basic working knowledge of the fundamentals of data communications, such as baud rate, parity, etc. If you don't, you may wish to consult a reference book, such as RS-232 Made Easy by Martin D. Seyer. You may also need to consult the documentation for your particular modem or RS-232 adapter.

You can specify the baud rate, parity, number of data bits, and number of stop bits desired for the T device before opening it. This can be done either using the TMODE utility program, or by setting values into memory directly from a program.

TMODE UTILITY

The TMODE utility is a PROMAL program provided on disk, which has the following command syntax:

```
TMODE [Baud [Parity [Databits [Stopbits ]]]]
```

If no arguments are given, it displays the current values. **Baud** is the desired baud rate, (default is 300 when PROMAL is started). Legal values can be 110, 300, 600, 1200, 2400, 4800, or 9600. You may also abbreviate 300 as 3, 9600 as 96, etc. When PROMAL is booted up, the initial baud rate is set to 300. The second optional argument is **Parity**, which should be specified as E, O, N, M, or S for even, odd, none, mark or space, respectively. The initial default is none. The third argument is **Databits** which should be 7 or 8. The initial default is 8. The final optional argument is **Stopbits**, which should be 1 or 2. The initial default is 1. Most systems use 1 stop bit except at 110 baud. Optional arguments which are not specified remain unchanged.

The values specified by the TMODE command will take effect the next time the T device is opened (or used in an EXECUTIVE command). The values set by TMODE may also be set directly from a program, discussed below.

PROGRAMMING THE T DEVICE

For many applications, programming the T device is very simple. You just need to open it and then input or output to it the same way you would a file. For example:

```
WORD MODEM  ; File handle for T device
BYTE BUFFER[81] ; Input buffer for T device
...
MODEM=OPEN("T",´B´)  ; Open T for input & output
IF MODEM=0  ; Trouble?
  ABORT "#CCan´t open T device"
PUTF MODEM, NL,"This is transmitted over the modem.",NL
...
IF GETLF(MODEM,BUFFER,80)
  PUT BUFFER,NL ; Display line received from modem
...
```

The primary added complexity of dealing with a modem is handling the situation where no data is received when it is expected. To handle this, a status routine is provided, to tell you when data is ready to read. In addition, global variables are provided to allow selecting different communications parameters under program control. These definitions are given in the file PROSYS.S and are summarized below:

```
EXT ASM FUNC BYTE GETTST AT $0FC6 ; TRUE if ready. Arg=0 input,1=output.

EXT BYTE TBAUD   AT $0DE9 ; 3=110,6=300,7=600,8=1200,A=2400,C=4800,E=9600
EXT BYTE TPARITY AT $0DEA ; 0=none, 1=odd, 2=even, 3=mark, 4=space
EXT BYTE TDATAB  AT $0DEB ; 0=8 bits, 1=7, 2=6, 3=5
EXT BYTE TSTOPB  AT $0DEC ; 0=1 stop bite, 1=2 stop bits
EXT BYTE TEOFCH  AT $0DED ; EOF char. (CTRL-Z default), unless TDEVRAW set
EXT BYTE TDEVALF AT $0DEE ; Auto line feed, $00=no, $80=out,$40=in,$C0=both
EXT BYTE TDEVRAW AT $0DEF ; "Raw" mode flag, $80 = no EOF or LF processing
EXT BYTE TDEVST  AT $0DF0 ; Status byte from last operation (see below)
```

Function GETTST requires one argument which is either 0 (to test the input status of the T device) or 1 (to test the output status). The function returns TRUE if the serial device is ready and FALSE otherwise. For input, it will return TRUE when at least one character has been received and can be read. For an example of how to use GETTST, see file TINYTERM.S.

The variables TBAUD, TPARITY, TDATAB, and TSTOPB can be used to set the same values which are set or displayed by TMODE, from within a program, for example:

```
TBAUD=$08
TPARITY=2
```

This selects 1200 baud with even parity. The desired values should be set prior to opening the T device. See the SRECEIVE.S and SSEND.S files for examples of how to set these variables.

TEOFCH is used to determine what character should be treated as End-of-File for input from the T device, defaulting to CTRL-Z ($1A). The default value allows a remote serial device to be used for input to the EXECUTIVE by redirecting input to the T device. TEOFCH is particulary significant for programs which use GETBLKF to read from the T device (generally not recommended).

In a program, you will often not want **any** character interpreted as end of file. This can be done by setting the TDEVRAW flag to $80 (not to TRUE!), which causes the T device to pass all characters straight through. The TDEVALF byte is the "auto line feed" flag. Setting bit 7 to 1 causes the T device driver to add a line feed ($0A) automatically after every CR ($0D) is sent. This may be needed it you have a serial printer or another computer connected to the serial port. Setting bit 6 of TDEVALF to 1 causes the driver to discard incoming linefeeds. If the TDEVRAW flag is $80, both TDEVALF and TEOF are ignored.

DETAILED INFORMATION FOR APPLE II T DEVICE

The Apple II T device driver supports the Apple Super Serial card and true compatible cards, and the Apple IIc serial port 2. For maximum flexibility, the PROMAL device driver manipulates the 6551 chip hardware directly. Therefore it does not support the Apple Pascal escape-sequences for selecting communication attributes, etc. (which are unsuitable for many applications). PROMAL does not support buffered T device input using interrupts, because of the incompatibility of some serial cards. This means that your application program may have difficulty "keeping up" with an incoming stream of characters from the T device at higher baud rates if it does extensive screen output or other time-consuming activities. Expert programmers with serious telecommunications applications may wish to write their own interrupt service routine, following the guidelines in the Apple Reference Manual. For this reason, or in order to support incompatible cards, PROMAL leaves "hooks" for writing your own T device drivers. If your serial card does not have a 6551 chip with its data register at $C0A8, you will have to write your own driver to use the T device.

The WORD at $0DF1 (Apple only!) is a pointer to a table of WORDs containing the addresses of the initialization, status, input, and output routines, respectively, used by the PROMAL T device. All are machine language routines. The INIT routine has no arguments and returns nothing. The STATUS routine expects A=0 for input or A=1 for output, returns the status in A, and the carry bit set if ready. The INPUT routine has no arguments and returns the character in A. The OUTPUT routine expects the character in A and returns nothing.

The TDEVST byte is set by any status, input, or output calls, as follows:

Bit 0 = Parity error	Bit 4 = Transmit buffer empty
Bit 1 = Framing error	Bit 5 = DCD not state
Bit 2 = Overrun error	Bit 6 = DSR not state
Bit 3 = Receive buffer full	Bit 7 = Interrupt flag

We have successfully used the T device on the Apple at the full 9600 baud with the built in drivers (for example, the SSEND and SRECEIVE programs). Naturally, this depends on your program. For example, if you attempt to access disk or have another time-consuming activity while characters are received, you will lose characters. In this case you should either arrange to have transmission halted temporarily (for example, using XON-XOFF protocol), or use a machine language buffered interrupt service routine.

DETAILED INFORMATION FOR COMMODORE 64 T DEVICE

The Commodore 64 uses the standard "Kernal" ROM support for RS-232, and is therefore subject to the same limitations. Opening the T device causes a 512 byte buffer to be allocated at LOFREE (an open error may indicate that there is not enough room for this), and LOFREE is moved up accordingly. This buffer is filled and emptied by the non-maskable interrupt routine in ROM. The RS-232 device is always opened in Commodore "3-line" mode; "X-line" is not supported due to problems in the Commodore firmware. For the Commodore, the T device driver will return an end-of-file indication on input if the buffer is empty or the Break detected bit is set in the status.

When using a modem (as opposed to direct connection through an RS-232 level shifter such as the Commodore 1011A), you will need to do additional programming to control the special modem functions. For example, to use the model 1660 300-baud modem, you will need to access the parallel port (user port) to go "off hook" **after** you open the T device. The TINYTERM program illustrates how to do this. For other modems or features such as dialing, you will need to consult your modem manual.

In general, we recommend you do not exceed 600 baud on the Commodore, although we have had success with 1200 baud provided that a long "burst" is not send to the Commodore at the full 120 characters per second. Naturally, this depends on the ability of your program to keep up.

The TDEVST byte reflects the status after any input, output, or status call to the T device, as follows:

Bit 0 = Not functional	Bit 4 = Not functional
Bit 1 = Framing error	Bit 5 = Not functional
Bit 2 = Receiver buffer overrun	Bit 6 = Not functional
Bit 3 = Receiver empty / Transmitter full	Bit 7 = Break detected

SSEND AND SRECEIVE PROGRAMS

The files SSEND.S and SRECEIVE.S on the PROMAL disk are source programs for transmitting files between computers with PROMAL, at speeds of up to 9600 baud on the Apple or 1200 baud on the Commodore 64. The files do not have to be text files; any kind of PROMAL file can be sent.

The programs provided form a complementary pair. SSEND transmits a specified file using an error-correcting protocol, and SRECEIVE receives the file on another computer and installs it on disk. The programs can be used to transfer any size file at up to 9600 baud between computers in close proximity without a modem, by using a simple "null modem" cable between serial ports, as shown below. If used with a Commodore 64, we suggest you limit transmission to 600 baud.

The diagram below illustrates how to wire a direct-connect cable (null modem), with pin connections for the Apple IIc 5 pin connector on port 2, or the Commodore 64 RS-232 adapter model 1011A or similar level shifter.

Computer A				Computer B		
Apple IIC DIN-5 Pin #	Commodore 64 1011A Pin #	RS-232 signal Name (#)		RS-232 signal Name (#)	Commodore 64 Port Pin #	Apple IIC DIN-5 Pin #
1	6	DTR (6)	-------	DSR (20)	20	5
2	3	TD (3)	-------	RD (2)	3	4
3	7	GND (7)	-------	GND (7)	7	3
4	2	RD (2)	-------	TD (2)	2	2
5	20	DSR (20)	-------	DTR (6)	6	1

Two remote computers can also exchange files at the maximum baud rate supported by the modem used (typically 300 or 1200 baud). When using a modem instead of a direct connection, some modifications may need to be made to the program to send modem control commands (such as to answer the phone). These modifications are entirely dependent on the type of modem being used. Consult your modem manual for further information.

To transmit a file, start the **receiving** computer's program first, for example:

```
SRECEIVE MYFILE.T 1200
```

will receive a file called MYFILE.T at 1200 baud. Then start the transmitting computer's program (at the same baud rate, of course):

```
SSEND MYFILE.T 1200
```

The file will be transmitted in 1K blocks with a verification "handshake" after each block is correctly received. If a block is garbled in transmission, the receiving program requests a retransmission of that block. The program exits when the entire file is received.

The source code for SSEND and SRECEIVE has comments which explain the operation of this simple communications protocol in detail. You may freely incorporate any part of these programs in your own projects.

TINYTERM

The TINYTERM program provided in source form on the PROMAL disk is a tiny terminal emulator program which provides the basic functions necessary to access a remote computer using an external modem. This program provides a "bare bones" communication program which can be used to communicate with many remote time sharing services, such as Compuserve. It is not intended to provide the functionality of commercially available communication packages, but is a simple program to illustrate the use of T device. Advanced users may enjoy enhancing it to a full communications package, perhaps adding the ability to upload and download disk files, etc. You may need to modify the program somewhat for use with your modem. We recommend 300 baud operation.

T DEVICE NOTES

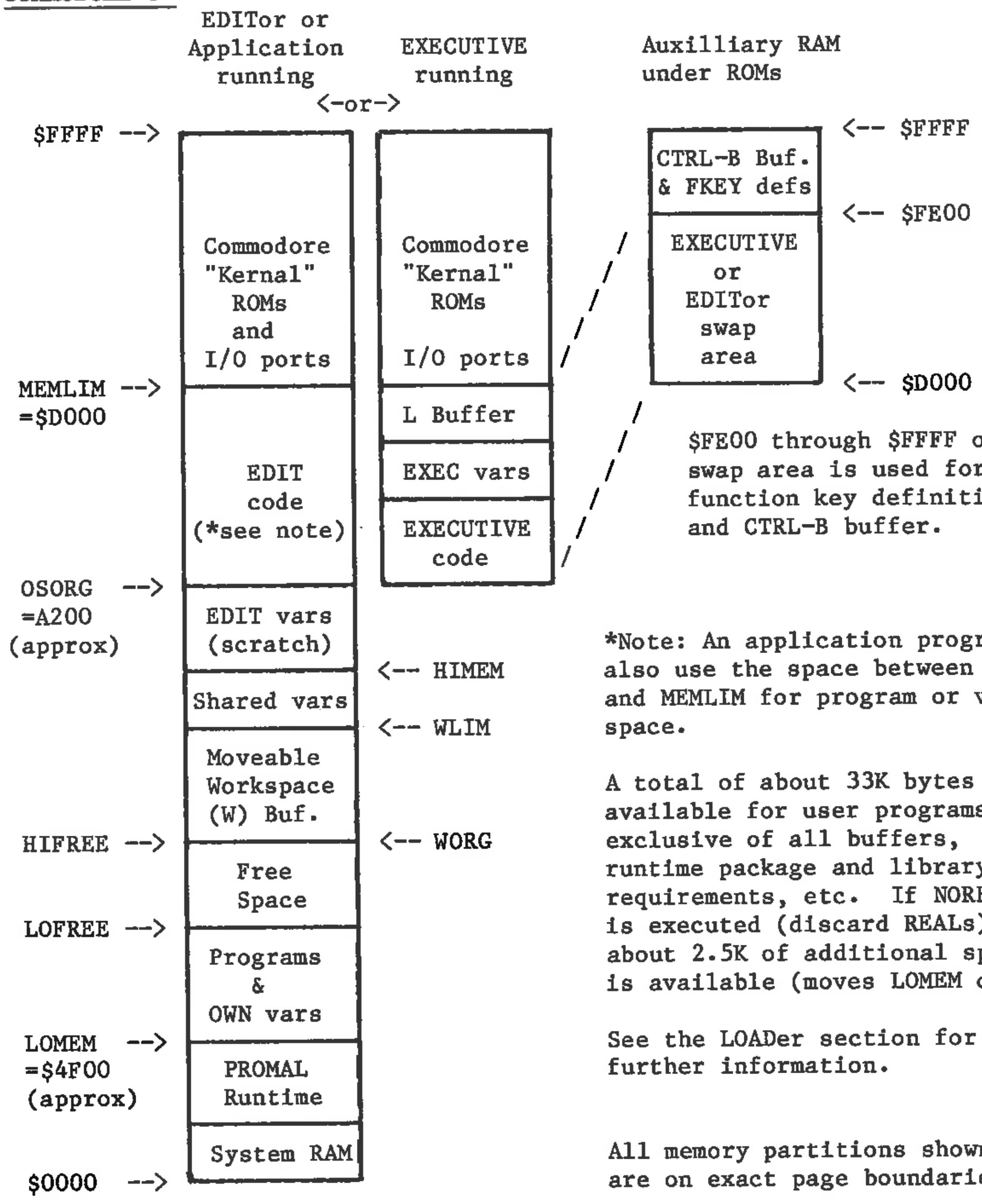
Data communications professionals know that programming serial data transmission is seldom as easy as it looks. It is important to understand that while the RS-232C document defines a "standard" way for computers to communicate, it is an extremely general standard with many possible variations. Not only can parameters such as baud rate and parity be selected in a wide variety of permutations, but there is no standard way for programs to address the modem itself. The port addresses for the modem are different on each computer, and are even different on various serial boards or attachments for the same computer. There is no agreement about what commands should be sent to the modem to make it perform its special functions (such as dialing or going "off hook"). Worse yet, there are a maddening variety of software "protocols" in use which govern the way information should be transferred between devices attached with a serial interface.

All these factors make it impossible to make a "one size fits all" driver for the T device. In implementing the T device, we have tried to make it easy to use for the vast majority of cases, and not impossible for the rest. It is entirely the responsibility of the programmer to insure proper data communications for any particular piece of communications equipment. This should be considered part of the application program.

APPENDIX G

MEMORY MAP

COMMODORE 64



$FE00 through $FFFF of the swap area is used for function key definitions and CTRL-B buffer.

*Note: An application program can also use the space between OSORG and MEMLIM for program or variable space.

A total of about 33K bytes is available for user programs, exclusive of all buffers, runtime package and library requirements, etc. If NOREALS is executed (discard REALs), about 2.5K of additional space is available (moves LOMEM down).

See the LOADer section for further information.

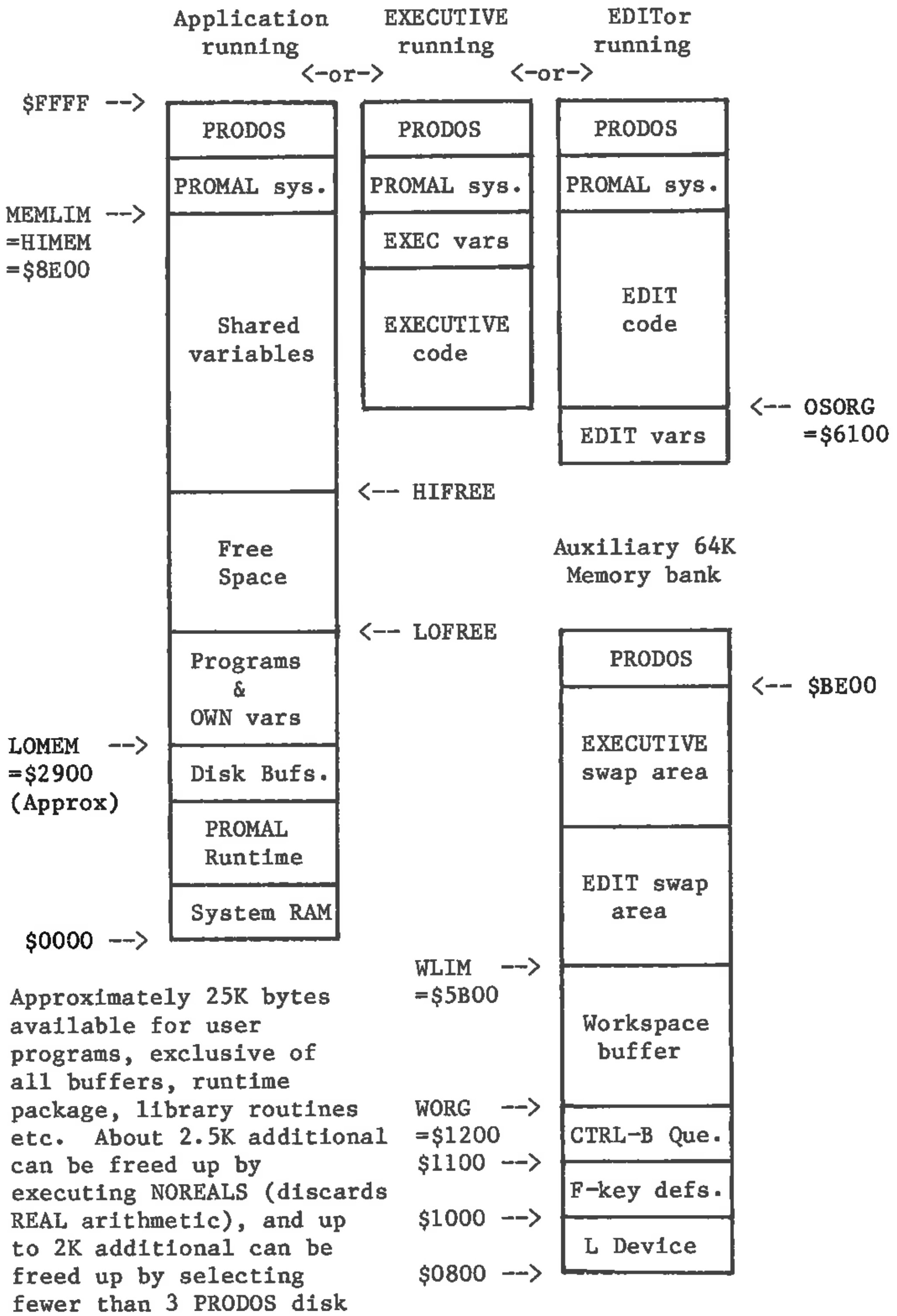
All memory partitions shown here are on exact page boundaries.

COMMODORE 64

Address	Description
0000 - 0001	6510 On-Chip I-O port
0002 - 0010	Available (Used by BASIC only)
0011 - 0015	Reserved for PROMAL enhancements
0016 - 0019	Used by PROMAL
001A - 002A	Used by C-64 Kernal
002B - 0042	Used by PROMAL
0043 - 0056	Used by C-64 Kernal
0057 - 0089	Used by PROMAL
008A - 00F2	Used by C-64 Kernal
00FB - 00FE	Available
00FF	Used by C-64 Kernal
0100 - 01FF	Hardware Stack
0200 - 0333	Used by C-64 Kernal
0334 - 03FF	Available for M/L programs (see note 2)
0400 - 07E7	Screen memory
07F8 - 07FF	Sprite data pointers
0800 - 08FF	Floating point stack
0900 - 09FF	Heap for Local Variables
0A00 - 0AFF	Scratchpad for I/O, encode/decode, etc.
0B00 - 0DFF	PROMAL System Data Area, see PROSYS.S for details. Reserved.
0E00 - 43FF	PROMAL Vectors, Jump Table, Nucleus, Library, & DYNODISK drivers.
4400 - 4EFF	PROMAL REAL processing routines, **or** Allocatable user memory.
4F00 - A1FF	(Approx.) Allocatable memory for user programs & workspace.
A200 - CFFF	(Approx.) EDITor/EXECUTIVE space, **or** user programs & variables.
D000 - FDFF	(Approx.) EDITor/EXECUTIVE swap area and L device (RAM)
FE00 - FFFF	Function key defs and CTRL-B buffer (RAM)
D000 - FFFF	C-64 Kernal ROMs, VIC, SID, and IO.

Notes:
1. All addresses subject to change without notice.
2. $0334 - $036F reserved for PROMAL hi-res graphics package.
3. File PROSYS.S contains definitions of many system locations.
4. See the Chapter 8 of the PROMAL LANGUAGE MANUAL and Appendix H for further information on memory allocation.

APPLE II



Approximately 25K bytes available for user programs, exclusive of all buffers, runtime package, library routines etc. About 2.5K additional can be freed up by executing NOREALS (discards REAL arithmetic), and up to 2K additional can be freed up by selecting fewer than 3 PRODOS disk buffers (BUFFERS command).

APPLE II

Address	Description
0000 - 000E	Available
000F - 0049	Apple II Monitor
004A - 004D	Available
004E - 0055	Apple II Monitor
0056 - 00AF	Used by PROMAL
00B0 - 00FF	Available
0100 - 01FF	Hardware Stack
0200 - 027F	Apple input buffer, used for scratch
0280 - 02BF	Apple input buffer; Scratch path name buffer for PROMAL
02C0 - 02FF	Apple input buffer, used for scratch
0300 - 03EF	Available (See note 2)
03F0 - 03FF	Apple II Vectors
0400 - 07FF	Text and low-resolution graphics display buffer
0800 - 08FF	Floating point stack
0900 - 09FF	Heap for local variables
0A00 - 0AFF	Scratchpad for I/O, encode/decode, etc.
0B00 - 0DFF	PROMAL system data area (see PROSYS.S file). Reserved.
0E00 - 10FF	(approx) PROMAL Vectors, Jump Tables, tables, etc.
1100 - 1BFF	(approx) PROMAL REAL processing, **or** Buffers / allocatable space
1C00 - 1CFF	(can vary) File descriptor table
1D00 - 28FF	(can vary) Disk buffers (3) for ProDOS
2900 - 8BFF	(can vary) Allocatable memory for user programs & variables
6100 - 8BFF	System space for EXECUTIVE & EDIT. Programs may overwrite.
8E00 - BEFF	PROMAL nucleus and library routines.
BF00 - BFFF	ProDOS page
C000 - FFFF	Apple I/O & system memory.

Auxiliary (bank-switched) memory:

0800 - 0FFF	Reserved for L device (library text).
1000 - 10FF	Reserved for function key strings.
1100 - 11FF	Reserved for CTRL-B buffer.
1200 - 5BFF	Workspace buffer.
6000 - BEFF	Swap area for EDIT and EXECUTIVE.

Notes:
1. All addresses subject to change without notice.
2. $0334 - $036F reserved for PROMAL hi-res graphics package.
3. File PROSYS.S contains many definitions of system locations.
4. NOREAL command causes allocatable space to start at 1E00 normally.
5. BUFFERS command can change allocatable space.
6. Developer´s version allows applications without auxiliary (bank switched) memory or 80 column card to run (see Developer´s guide).

IMPORTANT SYSTEM DATA AREA ADDRESSES

The following global variables are more precisely defined in the file PROSYS.S unless otherwise noted.

Address	Description
0BB0-0BBB	LDNAME - Command name of last LOAD attempt
0BC0	LDNOCHK - Flag, $80 if bypassing checksum check during loading
0C00-0C03	STDIN, STDOUT File handles (defined in LIBRARY.S file)
0C08	IOERROR - Error code from disk functions (LIBRARY.S)
0C0B	BFILTYP - System-dependent file type for OPEN.
0C0C	DIOERR - Disk I/O error, 0=ok,1=full,2=read err,3=wrt err.
0C0D	DFEXT - ´C´, default file extension for PROMAL files.
0C12-0C15	DATE - Day, Month, Year-1900, 1 byte each
0C16-0C17	LOMEM - Start of Allocatable Memory
0C18-0C19	LOFREE - Next available address for program load, $XX00
0C1A-0C1B	HIFREE - First address not allocatable for programs, $XX00
0C1C-0C1D	HIMEM - End of normally allocatable memory + 1, $XX00
0C1E	LDERR - Loader error return code, $00=success
0C1F	NLT - Number of loaded modules, including EDITor, EXEC.
0C22-0C23	RANDWD - Seed for random number generator (non-zero)
0C2B-0C2C	OSORG - Starting address for EDITor/EXECUTIVE, $XX00
0C2D-0C2E	MEMLIM - End of usable memory (if EDITOR discarded, C-64).
0C51-0C52	MLP - Address of subroutine called by PROC JSR
0C53-0C57	REGA, REGX, etc. - Registers for GO command or BRK
0CF2	NOFNCHK - Flag, if TRUE defeat default file extension
0CFF	BLINKD - Blink delay for cursor. >$7f=solid, 0=invisible.
0D00-0D50	CLINE - Current Command line, complete (LIBRARY.S)
0D51	NCARG - Number of arguments passed on command line (LIBRARY.S)
0D52-0D73	CARG - Array of pointers to arguments on comd. line (LIBRARY.S)
0D74-0DC4	COMD - Command line split into argument strings
0DC5-0DCC	WORG, WPTR, WEOF, WLIM - Pointers for Workspace (see also 0DDB)
0DCD-0DCE	LORG, LPTR, LEOF, LLIM - Pointers for Library (under ROMs, C-64)
0DDB-0DDC	WSIZE - Current Workspace size
0DDD-0DDE	GVORG - Address of start of all shared variables, $XX00
0DE9	TBAUD - T baud (3=110,6=300,7=600,8=1200,$A=2400,$C=4800,$E=9600)
0DEA	TPARITY - T device parity (0=none,1=odd,2=even,3=mark,4=space)
0DEB	TDATAB - T device data bits (0=8,1=7,2=6,3=5)
0DEC	TSTOPB - T device stop bits (0=2,1=2)
0DED	TEOFCH - T device end-of-file char for input (default=CTRL-Z)
0DEE	TDEVALF - T linefeed, bit 7=1=add on output, bit 6=1=strip on input
0DEF	TDEVRAW - T Raw mode flag, $80 = pass all chars through as is
0DF0	TDEVST - T device status for last operation, system dependent
0DF6	DRTERR - Copy of runtime error (See Developer´s Guide)
0DFD	PBLKCNT - # bytes actually written on last PUTBLKF
0DFF	ALPHALK - Keyboard Alpha lock flag, $80 = upper case only
1000	BKEYDEL - Key for delete with pullback
1001	BKEYINS - Key for begin insert mode
1002	BKEYJS - Key for jump to first char of line
1003	BKEYJE - Key for jump to last char of line
1004	BKEYCEL - Key for clear to end of line
1005	BKEYALK - Key for alpha lock toggle

1006	BKEYCAN	- Key for cancel line
1007	BKEYBT	- Key for backtrack prior line
1008	BKEYBS	- Key for backspace
1009	BKEYTAB	- Key for tab (indent)
100A	BKEYRT	- Key for cursor right
100B	BKEYLFT	- Key for cursor left
100C	BKEYFK1	- Key for first function key
100D	BKEYFKL	- Key for last function key
100E	BKEYEOF	- Key for E-O-F from keyboard

Note: All addresses subject to change without notice.

System addresses for **Commodore 64** only:

0C68-0C6A	PREFIX	- Current drive prefix string
0CFB	EDRES	- Flag, $80 if EDITOR is in memory, $00 if not
0DE0	C64DDV0	- C-64 disk device # for logical drive 0 (default=8)
0DE1	C64DDV1	- C-64 device # for drive 1 (9 for 1541, 8 for MSD)
0DE2	C64N1541	- Flag, $80 = permanently disable DYNODISK
0DE3	C64DYNO	- Flag, $80 = DYNO on , $00 = DYNODISK off
0DF3	C64PSA	- Secondary Address for Printer OPEN (See Appendix D)
0DF4	C64PUL	- Printer upper/lower case switch flag (See Appendix D)
0DF5	C64PDV	- Device number for printer (See Appendix D)

System addresses for **Apple II** only:

0C68-0CA3	PREFIX	- Current volume & pathname (ends with '/')
0DE0	ABORTCH	- Program abort character (default=CTRL-C, $00=none)
0DE5	RAMUNIT	- /RAM unit number, normally $B0=slot 3 drive 2
0DE6	DSLOT	- Slot for 1:, 2: drive, normally 6
0DF1-0DF2	TDEVTBL	- Address of T device driver table (see Appendix F)
0DF3	APLPALF	- Auto line feed flag for printer ($80=yes, $00=no)
0DF4-0DF5	APLPJT	- Pointer to printer driver vector table (points to Init ptr word, Output ptr word (A=char)
0DFB-0DFE	DSKBUFS	- Pointer to disk buffers.

Notes:

1. All addresses subject to change without notice.
2. File PROSYS.S contains the definitions of these and other system locations.