KGN COMAL
USER MANUAL

Author : Hans Otten
Version : 1.0 March 2025

Preface

KGN COMAL is an interpreter for the COMAL structured language.

The Dutch KIM Gebruikers Club (KGN) distributed a version of COMAL for the Elektor Junior. Later this version was enhanced for the DOS65 system to Version 2.1 with disk file I/O and video support. A KIM-1 version is not known.

In around 2015 several DOS65 and Elektor Junior computers were acquired. With it came a binary and paper document of KGN COMAL: a COMAL DOS65 binary V2.1 and a compact manual. On a Junior tape a binary of the first version of KGN COMAL was found.

In the Dutch KIM Kenner, the magazine published by the KGN club, two articles were published: a Maze program in KGN COMAL and a small change to KGN COMAL v2.1, both written by Antoine Megens.

Based upon the binaries a KIM-1 version is constructed in 2025. A partial disassembly of the I/O parts and debugging led to a working KGN COMAL for the KIM-1. Finding the Junior dependencies was the main job: the Junior routines OUTCH, GETCH and CRLF save X and Y registers, the KIM-1 is less caring.

This user manual was written in 2025, partially based upon the very compact Dutch manual from 1987 and the knowledge acquired during the port.

KGN COMAL has not been thoroughly tested yet on real hardware, the KIM-1 Simulator was a great help during the port. Screenshots in this manual are from the Simulator.

Contents

1 Installation and First Steps

KGN COMAL requires a Kim-1 or Junior with at least 16K RAM at $2000, and a (very simple) terminal.

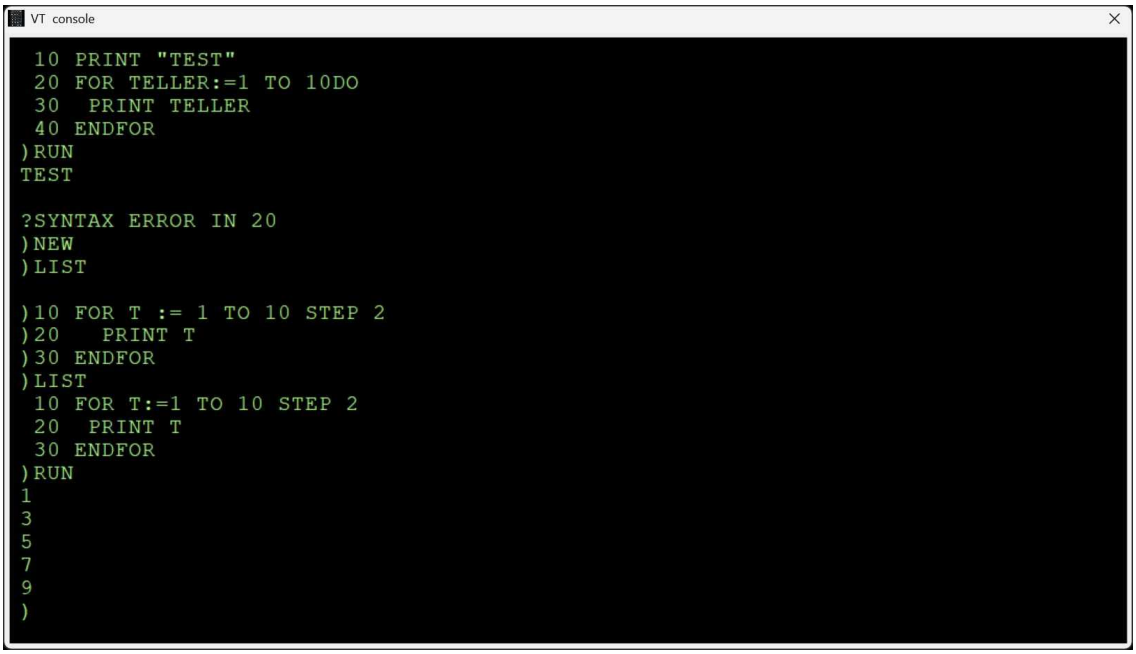
KGN COMAL is distributed in a ZIP archive, with folders for the applicable systems:
KGNCOMAL.ZIP

KGN COMAL Elektor Junior and KIM-1 are loaded from papertape: KGNCOMALKIM1.pap etc.
Load address is $2000, end address $47FF

Cold Start KGN COMAL at $3000. You will see a ‘)’ as prompt.

When COMAL is not responding you can press RESET or STOP.
A warm start is at $0000, the program is kept safe then. Type CLEAR to restore the housekeeping of COMAL.

You can use Direct mode: enter valid COMAL commands and press Enter.
Or you can use COMAL to execute programs consisting of multiple lines with one COMAL statement per line via a RUN.



2 Saving and loading of COMAL programs

The Junior version of KGN COMAL has audio cassette tape Save and Load facilities.
Use is made of the Junior ROM routines, which are callable as subroutines.

For the KIM-1 version a not perfect solution is implemented, that requires use of the KIM-1 monitor to save and load. See the KIM-1 chapter for that.

Nowadays using audio tape for saving and loading is not very practical. It is more convenient to use the terminal emulator logging and replay facilities for text.
Teraterm for example has Log and Replay options to capture text coming from the KIM-1 via the serial output etc and replay text from a text file into the KIM-1 serial input.

Save a COMAL program

Start logging in the terminal emulator, in COMAL LIST the program, and then close the logging.
Remove the unnecessary text from top and bottom of the text file and you have a perfect listing on the PC as typed in on the KIM-1.

Load a COMAL program

The text file created by the Save method above is perfect to replay to COMAL.
But COMAL and the KIM-1 require some work on the text file and communication:

1. The text file needs to have CR line ends, not CRLF as DOS/Windows or LF as Unix/Linux.
   This can be set for example with Notepad++ in Edit – EOL conversion.
2. The text file needs to be an ANSI text file (8 bit ASCII characters).
   This can be set for example with Notepad++ in Encoding – ANSI
3. The terminal emulator needs to give the KIM-1 with COMAL some time to handle incoming characters and Returns.
   Set for example in Teraterm Line delay to 200 ms and Character delay to 20 ms.

3 Syntax rules

Uppercase
All text in a COMAL program is in uppercase, with the exception of characters in a string.

Variable names
A name is maximum 2 characters long. Starts with A-Z, second character A..Z or 0..9.
String variables have a '$' attached.

Examples:
Real and integer A, B1, CC
String S, B1$

Data types

KGN COMAL has three basic datatypes: Integer, Real and String.

Integers are whole numbers, stored in 5 bytes and so can be very large.
Reals are floating point numbers with 9 digit precision.
A string is an array of characters, maximum 255 bytes long, with ANSI characters code 0-255.

Besides these basic datatypes KGN COMAL also knows multidimensional integer arrays, see the DIM statement.

Booleans
A true is represented by the number 1, a false by number zero.

Expressions
<expression> can be a <numeric expression> or a <String expression>
A <numeric expression returns a numeric value, integer or real.
A <string expression> returns a string>

<numeric constant> is a decimal representation of a number.
<numeric string constant> is a string of characters enclosed in double quotes.
Examples
Integer A := 1
Real R := 0.5
String S := "This is a string"

Variables are implicit created when they are found in a program, no explicit declaration.
The type is determined by assignment. COMAL is very tolerant to mixing reals and integers, conversion and rounding takes place automatically.

Example
)LIST

```
 10 A:=32767
 20 B:=49999
 30 C:=100000
 40 PRINT A;" ";B
 50 PRINT C
 60 D:=10000
 70 PRINT D
 80 D:=10*D
 90 PRINT D
 100 D:=10*D
 110 PRINT D
 120 D:=D*10
130 PRINT D
)RUN
32767 49999
100000
100000
1000000
10000000
100000000
```

Program lines and Editing.

Each COMAL statement in a program needs to be a line, one at a time.
Each line has a <line number>, which are only for entering and listing the program.
A line is entered by typing the line number followed by the COMAL command, and entered via Return. The only line editing is using BACKSPACE to erase the last entered characters.
A line can be removed by typing just the line number.
See the Direct Commands: LIST, DELETE, RENUMBER and LIST for more information.

<line number> is an integer in the range 1-63999

Constants

Numeric constants can be entered as a string of numbers or in scientific notation:
200000 and 2E+5 are identical.

A string constant is a string of characters enclosed in double quotes.
Only in strings lower case (and any control character) is allowed.
“This is a valid COMAL string”

4 Direct commands

Most COMAL statements can be used in direct mode, one at a line.

Direct commands, not to be used in a program, for managing the program lines are:

DELETE
DELETE <line number 1> - <line number 2>
Deletes line from the program.

RENUMBER
RENUMBER [<line number>[, <step>]]
Renumbers lines. The first line becomes <line number>, each following line number is incremented in incremented by <step>.
IF <step> is not specified the step increment is 10.
If <line number> is not specified the first line is 10, and the increment 10.

LIST
Prints the program.
LIST without arguments shows the whole program.
LIST[<line number 1>][– <line number 2>]
LIST <line number 1> shows the program from <line number 1> till the end.
LIST - <line number 2> shows the program from the <line number 1> till <line number 2>

CLEAR
Remove all variables and does a RESTORE.

NEW
Removes the whole program and clears all variables.

See also the RUN, CONT, command.

5 Operators

Operators are part of expressions.
<expression 1 > <operator> <expression 2>

There are two types of expressions in KGN COMAL:
<numeric expressions> contain constants, variables, numeric functions, used with parentheses.
<string expression> contain string of characters.

Sign
+
-

Numeric

+ sum
- subtract
^ power off
* multiplication
/ division
< less than
> greater than

Logical <expression1> <logical operator> <expression2>

AND result 1 if both of the expressions not equal to zero
OR result 1 if one of the expressions not equal to zero

String
+ concatination

6 Functions

Result := <function(<argument>)

Arithmetic functions

NOT
Result is 1 if argument not equal zero

SGN
-1 for negative argument, +1 for positive argument

INT
Integer value of real argument

ABS
Absolute value of argument

FRE
Performs garbage collect and result is number of free memory locations
Note that the number may be negative for large amount of RAM. Add 65536 to get the real free memory.

POS
Current position in the line of output

SQR
Square root

RND
If argument < 0 the the result is a number between 0 and 1
If argument > 0 the the result is a random number between 0 and 1
If argument = 0 the the result is the previous random number

LOG
Natural logarithm

EXP
e to the power of argument

COS
Cosinus of argument in radials

SIN
Sinus of argument in radials

TAN
Tangent of argument in radials

ATAN
Arc Tangent of argument in radials

String functions

LEN
Length of string

STR$
Converts numeric argument to string

VAL
Converts string to number. Read from the start of the string until non-numeric

ASC
Result is ASCII code of first character of string argument

CHR$
result is string of 1 character with ASCII code of the numeric argument

LEFT$
LEFT$(string argument, number argument)
Result is string with (number argument) characters from the left of string argument

RIGHT$
RIGHT$(string argument, number argument)
Result is string with (number argument) characters from the right of string argument

MID$
MID$(string argument, number argument 1, number argument 2)
Delivers a string with number argument 1 characters starting at number argument 2.

Example

```
)10 S$ := “1234567890”
)20 M$ := MID$(s$,3,5)
)30 PRINT M$
)RUN
34567
```

I/O functions

USR
The number argument is placed in the floating point accumulator.
A CALL 10 is executed. On location 10 a JMP to a user supplied machine routine is to be placed.
The result of the user supplied routine has to be returned in the floating point accumulator.
An RTS returns to COMAL.
Note: the location of the floating point accumulator is unknown!

PEEK
The result is the content of the memory location at <number argument>.

7 Statements

Assignment
<variable> **:=** <expression>
Only <String expressions> can be assigned to <string variables>
See the conversion functions like VAL and ASC for type conversion.
COMAL is very tolerant when numeric types, reals and integers are concerned, you can mix them freely. Reals are rounded automatically to integers.

//
A line beginning with // is ignored by COMAL and can be used as comment.

PRINT
PRINT [,;[<argument 1>[,;]argument 2] ...[.;]

Prints the arguments on the terminal. Any expression is valid.
A comma will tabulate the output in multiples of 16 characters.
A semicolon at the end prevents the CRLF that otherwise ends the PRINT.

A '?' is also interpreted as PRINT.

```
Example:

)PRINT;10
10
)PRINT ,10
                10
)PRINT ;10,I
10              10
)PRINT I
10
)PRINT I;I
1010
)PRINT I,I
10              10
)PRINT ,I,I
                10              10
)PRINT "Text ",I
Text            10
)PRINT I/3
3.33333333
```

TAB
TAB(<arithmetic expression>)
Prints spaces to position <arithmetic expression>.
Only usable in a PRINT statement,

SPC
SPC(arithmetic expression>
Prints <arithmetic expression>spaces.
Only usable in a PRINT statement,

LABEL:
LABEL: <string expression>
A label on a line is a line where you can jump to with a GOTO.

GOTO
GOTO <label>

ONERR GOTO
ONERR GOTO <label>
If an error occurs during a program control is resumed at the ON ERR GOTO line, if found.
Here you can handle the error. End this routine with RESUME.
In memory location 222 you find the error code. In memory location 218 (low byte) and 219 (high byte) you find the line number.
In the following list you find the error belonging to an error code.

```
  0 NEXT WITHOUT FOR
 16 SYNTAX
 22 ENDPROC WITHOUT EXEC
 42 OUT OF DATA
 53 ILLEGAL QUANTITY
 69 OVERFLOW
 77 OUT OF MEMORY
 90 UNDEF' PROCEDURE
107 BAD SUBSCRIPT
120 REDIM'D ARRAY
133 DIVISION BY ZERO
163 TYPE MISMATCH
176 STRING TOO LONG
191 BAD FLOW OF CONTROL
224 UNDEF'D FUNCTION
225 BREAK INTERRUPT
```

ONERR can be switched off with a POKE 216,0

RESUME
Use this at the end of an ONERR routine. Program execution will resume at the line the error occurred.

RUN
RUN [<label>
Program execution starts at the line with LABEL : <label>>
If no label is specified execution starts at the first line of the program.
Note that RUN first does perform a ‘compilation’ of the program to optimize it.. If execution stops abnormally the listing is damaged. Enter the command RESTORE to repair the listing.

STOP
Causes a BREAK in the program. Use END. to repair the listing. Use CONT to resume the program.

CONT
Resumes program execution after a STOP or a BREAK.

END.
Stops the program and restores the listing. Use END. to prevent program execution going into PROC statements.

WAIT
WAIT <arithmetic expression>
Causes a delay. A numer of 125 is about 1 second.

DEF FN
DEF FN <variable 1> [variable 2 .. n>]

INPUT
INPUT <string>
Reads a string from the console and stores it in <string> until a Return is entered.
To read a number use this:
INPUT S$
A := VAL(S$).

CTRL-C followed by Return causus a BREAK, resume with CONT.

GET
GET <string>
Reads form the console one character, no RETURN required.
ctrl-c BREAKS.

DIM
DIM <var>(<arithmetic expression 1> [..<arithmetic expression N>
Defines a multidimensial array of numbers with name <var>.
<arithmetic expression N> defines the number of elements +1 in dimension N.
A dimension can not exceed 32767 elements.
An array is referred to as:
<var>(number, number ...).
An array is zero based!

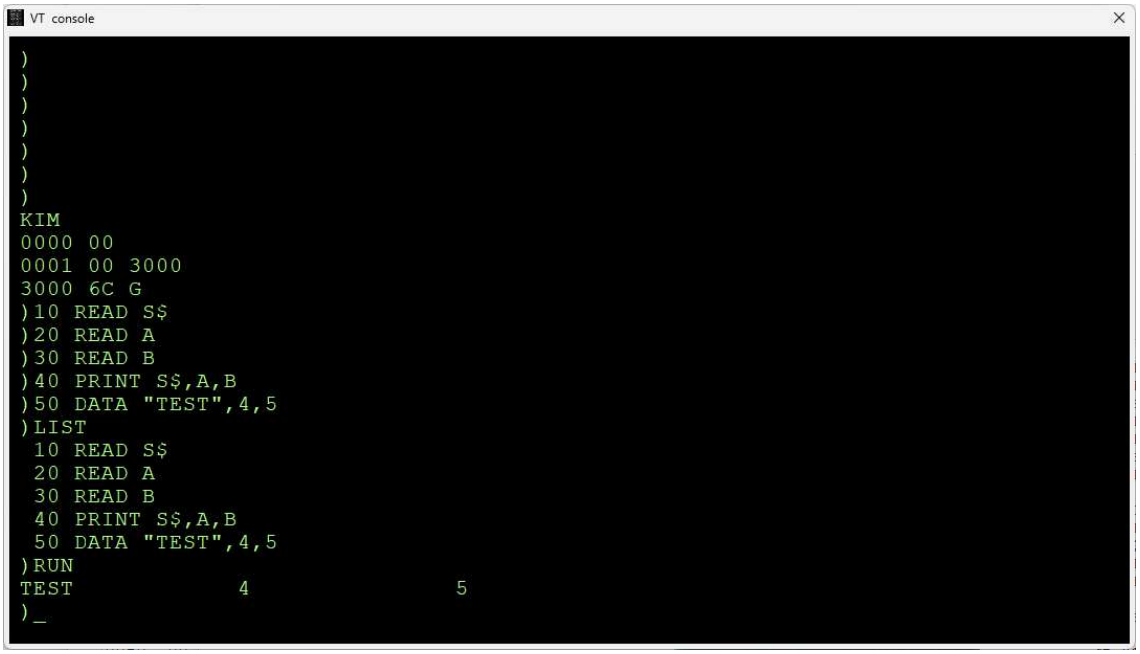
Example

```
)10 DIM AR(10,10)
)20 AR(10,10) := 5
)30 AR(0,0) := 4
)40 PRINT AR(10,10)
)50 PRINT AR(0,0)
)RUN
5
4
```

DATA
DATA constant [.. constant]
Local data in a program. Can be read with READ.
Constant can be a number or a string

READ
READ <variable>
Reads the next constant into <variable> from the DATA.

RESTORE
Allows to read the DATA from the beginning.



POKE
POKE <memory location>, <value>
Stores the <value> (0.255) into <memory location> 0 .. 65535

CALL
CALL <memory location>
Executes a machine language routine at <memory location (0..65535) >
Return to COMAL via RTS.

RES
RES(<procedure>, <var1> .. <varN>
Executes <procedure>, the result is varN. varN can not be a string.

LOAD
LOAD, <tape ID>
Loads a COMAL program from audio tape, <tape ID> is the tape ID used by the Junior as filename.
A tape ID is 1 to 254.

SAVE
SAVE, <tape ID>
Savess a COMAL program from audio tape, <tape ID> is the tape ID used by the Junior as filename.
A tape ID is 1 to 254.

8 Procedures

A procedure is a subroutine with a name.
A procedure is called with EXEC.

<block of statements> is one or more program lines.

PROC – ENDPROC
PROC <string expression>, [var1 ...[varN]]
  <block of statements>
ENDPROC

EXEC
EXEC <string expression>, [var1 ...[varN]]

<string expression>, usually a string constant.
[var1 ...[varN]] is a list of variables that are copied to variables in the PROCedure.
The local variables are copied back to the calling variables.

Note that PROCedures should not be reached at program execution. Place them at the end of a program and end the program before the PROC with END.

Example of a PROCEDURE:

```
 10 X:=1
 20 Y:=2
 30 S:=0
 40 PRINT "S= ";S
 50 EXEC: "ADD",X,Y,Z
 60 PRINT "S AFTER ADD = ";S
 70 END.
 100 PROC "ADD",X,Y,S
 110  S:=X+Y
 120 ENDPROC
)RUN

S= 0
S AFTER ADD = 3
)
```

9 Flow of control

<block of statements> is one or more program lines.
<boolean expression> is an arithmetic expression that delivers a Boolean result, TRUE (not zero) or FALSE (zero).

IF-THEN-ELSE-ENDIF
IF <boolean expression> THEN
  <block of statements 1>
[ELSE
 <block of statements 2>]
ENDIF

If <boolean expression is not zero (TRUE) <block of statements 1> is executed and the program continues after the line with ENDIF.
The ELSE part is executed if <arithmetic expression is zero (FALSE).
ELSE is optional.

WHILE-DO:-ENDWHILE
WHILE <boolean expression> DO
  <block of statements>
ENDWHILE

If <boolean expression is not zero (TRUE) <block of statements 1> is executed and the program continues, in a loop, at the WHILE statement.
The loop is executed 0 or more times.

REPEAT-UNTIL
REPEAT
 <block of statements>
UNTIL <boolean expression>

The block of statements is executed.
If <boolean expression> is FALSE the program continues in a loop at REPEAT, else at the next line.
The loop is executed 1 or more times.

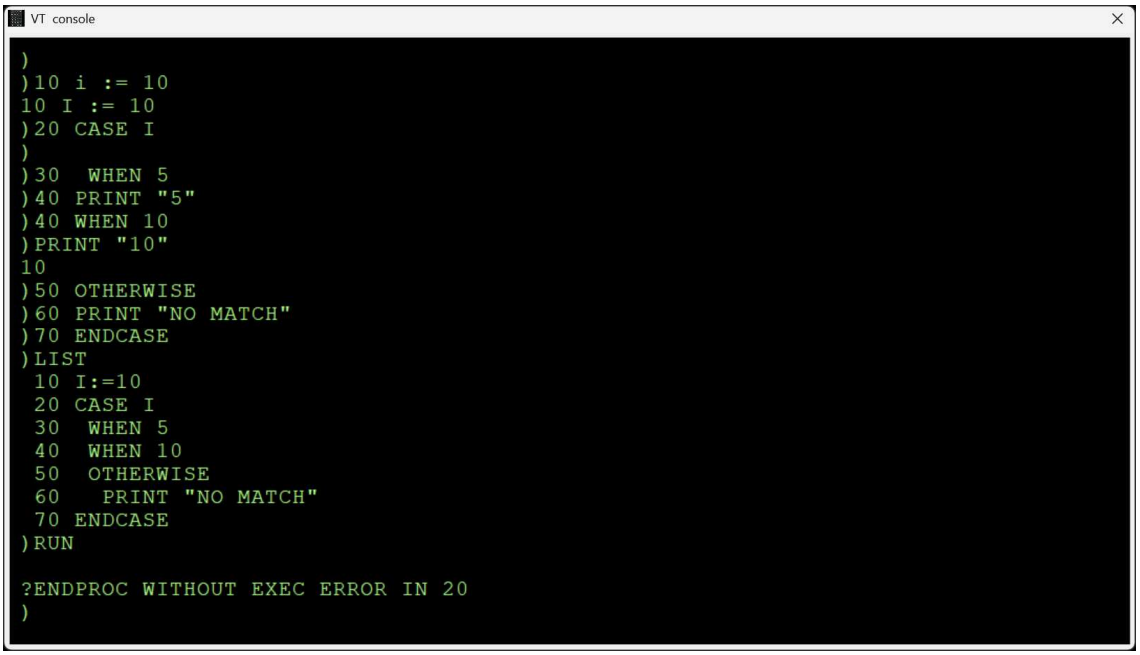
FOR-TO-STEP-ENDFOR
FOR <variable> := <arithmetic expr 1> TO <arithmetic expr 2> [STEP <arithmetic expr 3>]
  <block of statements>
ENDFOR

Variable get the value of <arithmetic expr 1>. If the variable is less than arithmetic expr 1 the block of statements is executed, the variable incremented with the step <arithmetic expr 3> and the program repeats the block until the variable is greater than the <arithmetic expr 1>.
STEP is optional, if omitted the step increment is 1.

CASE .. ENDCASE

CASE var
  WHEN value[..value]
    <block of statements>
  [WHEN value[..value]
    <block of statements>]
     .
     .
  OTHERWISE
    <block of statements>
ENDCASE

This construct seems not to work on KGN COMAL, Error message ENDPROC WITHOUT EXEC



10 Internals

Memory layout:

Zeropage usage
$0000 - $000D
$0067 - $0080
$00AF - $00CC

Keyboard buffer
$0200-$02FF

Memory above $2000
$2000 - $47FF COMAL interpreter
$4800 – highest RAM address for COMAL program and variables

COMAL searches with a non-destructive search for the highest RAM address at Cold start.

The first free location is stored at address $AF (low) and $B0 (high).
The highest RAM address + 1 is stored at $73, 74

To limit COMAL to a lower upper RAM address change this in COMAL, for example with upper address $8000

```
4195 A0 00          LDY #$00
4197 A9 80          LDA #$80
```

11 KGN COMAL Elektor Junior

The KGN COMAL for JUNIOR has partially been disassembled to aid the port to KIM-1.

The tape load/save routines extract from this disassembly is interesting:

```
;
; Load/save routines         JUNIOR KGN COMAL
;
28B0   20 D0 28             JSR L28D0  ; dos65 rewritten
28B3   A5 67                LDA $67    ; start of memory
28B5   8D 70 1A             STA $1A70  ; SAL JUNIOR
28B8   A5 68                LDA $68
28BA   8D 71 1A             STA $1A71  ; SAH JUNIOR
28BD   A5 69                LDA $69    ; end of memory
28BF   8D 72 1A             STA $1A72  ; EAL JUNIOR
28C2   A5 6A                LDA $6A    ;
28C4   8D 73 1A             STA $1A73  ; EAH JUNIOR
28C7   20 76 14             JSR $1476  ; dump to tape and print 'READY'
28CA   B0 01                BCS L28CD  ; something went wrong?
28CC   60                   RTS
28CD   4C 2A 24   L28CD     JMP L242A
;
; get ID, must be 01-FE
;
28D0   20 F5 28   L28D0     JSR L28F5  ; read a number
28D3   E0 00                CPX #$00
28D5   F0 F6                BEQ L28CD  ; error
28D7   E0 FF                CPX #$FF
28D9   F0 F2                BEQ L28CD  ; error
28DB   8E 79 1A             STX $1A79  ; store at Junior tape ID
28DE   60                   RTS
;
; load routine from tape
;
28DF   20 D0 28             JSR L28D0
28E2   20 96 14             JSR $1496  ; call Junior and print 'READY'
28E5   B0 E6                BCS L28CD  ; error
28E7   A5 FA                LDA $FA    ; start address to Comal JUNIOR
28E9   85 69                STA $69
28EB   A5 FB                LDA $FB    ; JUNIOR
28ED   85 6A                STA $6A
28EF   20 6C 26             JSR L266C  ; find end of program loaded
28F2   4C 3C 24             JMP L243C
```

12 KGN COMAL DOS65

DOS65 is a disk operating system. KGN COMAL has been ported to a DOS65 program.
The COMAL interpreter is the same as the Junior of KIM-1 version with some extras:

DOS65 video statements

INVERSE
CLS
ON
OFF

File I/O statements

DOS
CREATE
OPEN
CLOSE
DEL
CHAIN

COMAL statement

AUTO

13 KGN COMAL KIM-1

These are the adaptations made to KGN COMAL JUNIOR
The tape I/O routines have been patched so that the KIM-1 LOADT and SAVET is called.
This drops back to the KIM Monitor. See the source what to do and do a Warm start at $0000.

```
; KIM-1 Comal patch routines
;
; Hans Otten, March 2025
;
;
; KIM-1 ROM and 6530 addresses
;

SAD     =     $1740             ; 6530 A DATA
PADD    =     $1741             ; 6530 A DATA DIRECTION
SBD     =     $1742             ; 6530 B DATA
PBDD    =     $1743             ; 6530 B DATA DIRECTION
CLK1T   =     $1744             ; DIV BY 1 TIME
CLK8T   =     $1745             ; DIV BY 8 TIME
CLK64T  =     $1746             ; DIV BY 64 TIME
CLKKT   =     $1747             ; DIV BY 1024 TIME
CLKRDI  =     $1747             ; READ TIME OUT BIT
CLKRDT  =     $1746             ; READ TIME
;       ** MPU REG.  SAVX AREA IN PAGE 0 **
PCL     =     $EF               ; PROGRAM CNT LOW
PCH     =     $F0       ; PROGRAM CNT HI
PREG    =     $F1               ; CURRENT STATUS REG
SPUSER  =     $F2               ; CURRENT STACK POINTER
ACC     =     $F3               ; ACCUMULATOR
YREG    =     $F4               ; Y INDEX
XREG    =     $F5               ; X INDEX
;       ** KIM FIXED AREA IN PAGE 0  **
CHKHI   =     $F6
CHKSUM  =     $F7
INL     =     $F8               ; INPUT BUFFER
INH     =     $F9               ; INPUT BUFFER
POINTL  =     $FA               ; LSB OF OPEN CELL
POINTH  =     $FB               ; MSB OF OPEN CELL
TEMP    =     $FC
TMPX    =     $FD
CHAR    =     $FE
MODE    =     $FF
;       ** KIM FIXED AREA IN PAGE 23 **
CHKL    =     $17E7
CHKH    =     $17E8             ; CHKSUM
SAVX    =     $17E9             ; (3-BYTES)
VEB     =     $17EC             ; VOLATILE EXEC BLOCK (6-B)
CNTL30  =     $17F2             ; TTY DELAY
CNTH30  =     $17F3             ; TTY DELAY
TIMH    =     $17F4
SAL     =     $17F5             ; LOW STARTING ADDRESS
SAH     =     $17F6             ; HI STARTING ADDRESS
EAL     =     $17F7             ; LOW ENDING ADDRESS
EAH     =     $17F8             ; HI ENDING ADDRESS
```

```
ID      =     $17F9             ; TAPE PROGRAM ID NUMBER
;       ** INTERRUPT VECTORS **
NMIV    =     $17FA             ; STOP VECTOR (STOP=1C00)
RSTV    =     $17FC             ; RST VECTOR
IRQV    =     $17FE             ; IRQ VECTOR (BRK=1C00)
;
; KIM-1 ROM addressses
;
RST     = $1C22                 ; hardware reset
START   = $1C4F                 ; start KIM-1 processor, KDB selection
GETCH   = $1E5A                 ; GETCH (serial, with hardware echo)
OUTCH   = $1EA0                 ; OUTCH (serial)
CRLF   = $1E2F

              .org  $2553
             JSR    KGETCH

       .org   $28B0
              JSR L28D0    ; get tape ID
              LDA $67      ; start of memory
              STA $17F5    ; SAL KIM-1
              LDA $68
              STA $17F6    ; SAH KIM-1
              LDA $69      ; end of memory
              STA $17F7    ; EAL KIM-1
              LDA $6A      ;
              STA $17F8    ; EAH KIM-1
              JMP $1800    ; dump to tape and return to monitor
              BCS L28CD    ; something went wrong?
              RTS
L28CD         JMP $242A
;
; get ID, must be 01-FE
;
       JSR $28F5    ; read a number
       CPX #$00
       BEQ L28CD    ; error
       CPX #$FF
       BEQ L28CD    ; error
       STX $17F9    ; store at KIM-1 tape ID
       RTS
;
; load routine from tape
;
L28D0         JSR L28D0           ; get tape ID
              JMP $1873           ; load from tape and return to KIM monitor
              BCS L28CD           ; manually with KIM-1 monitor:
              LDA $FA             ; 17ED VEB + 1 to $69
              STA $69
              LDA $FB             ; 17EE VEB +2 to $6A
              STA $6A             ; G 28EF  from KIM monitor
              JSR $266C           ; find end of program loaded
                                    G here from KIM monitor
              JMP $243C
```

```
       .org   $2C11
              JSRKGETCH

       .org   $4248
              BIT    SAD

       .org   $424D
              BIT    SAD

       .org   $425C
       ;      STA    IRQV

       .org   $4261
       ;      STA    IRQV+1

       .org   $47FA
              JSR    KOUTCH

       .org   $47F5
              JSR    KCRLF

       .org   $4275
KGETCH        STX $4391
              STY $4392
              JSR GETCH
              LDX $4391
              LDY $4392
              RTS

KOUTCH        STA $4393
              STX $4394
              STY $4395
              JSR OUTCH
              LDA $4393
              LDX $4394
              LDY $4395
              RTS

KCRLF         STX $4391
              STY $4392
              JSR CRLF
              LDX $4391
              LDY $4392
              RTS
       .end
```

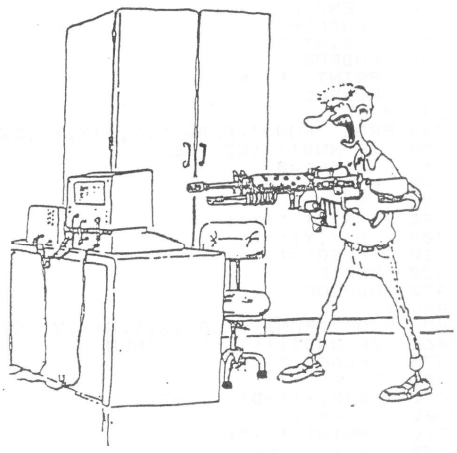
Appendix A DOS565 V2.1

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

;-----------------------------------------------------------
;       Changes made in COMAL version 2.1 for DOS65
;===========================================================
;       Antoine Megens                          May 1987
;-----------------------------------------------------------
;Directions:
;Load  old COMAL with LOAD S:COMAL,  then  enter  MONITOR.
;Change  the following addresses and save new COMAL  with:
;SAVE  S:NCOMAL  2000,4DFF,3000.  Then  enter  SETMODE  -C
;S:NCOMAL and test the changes with NCOMAL. The RUBOUT key
;should  work  on screen now and when the following  COMAL
;program is executed,  the file TEST.DAT should be  closed
;with  a  $00 byte instead of $1C (check this  with  DOS65
;command DUMP TEST.DAT).
;
; 100 CREATE "TEST.DAT"
; 110 OPEN #1;"TEST.DAT"
; 120 PRINT #1;"Testing EOF change"
; 130 CLOSE #1
;
; If this works you may rename the file NCOMAL to COMAL.
;-----------------------------------------------------------
28D2  4C D0 4D          JMP  $4DD0     ;was JMP $C023,
;                                      ;now check RUBOUT
;                                      ;first
4C94  A9 00             LDA  #$00      ;was LDA #$1C
;
; RUBOUT check routine (unused space in old COMAL)
;
4DD0  C5 2D             CMP  $2D       ;RUBOUT character?
4DD2  F0 03             BEQ  $4DD7     ;yes! else
4DD4  4C 23 C0          JMP  $C023     ;just print char.
4DD7  8A                TXA            ;at zero position
                                       ;of input?
4DD8  10 03             BPL  $4DDD     ;no, continue
4DDA  A5 2D             LDA  $2D       ;else exit with
                                       ;RUBOUT char.
4DDC  60                RTS            ;(do nothing)
4DDD  A9 08             LDA  #$08      ;print <BS><space>
                                       ;<BS>
4DDF  20 23 C0          JSR  $C023     ;to simulate
                                       ;RUBOUT on screen
4DE2  A9 20             LDA  #$20
4DE4  20 23 C0          JSR  $C023
4DE7  A9 08             LDA  #$08
4DE9  20 23 C0          JSR  $C023
4DEC  A5 2D             LDA  $2D       ;exit with RUBOUT
                                       ;char. in A
4DEE  60                RTS
```

Appendix B Amazing Maze COMAL program


```
1000 //  ***************************
1010 //  * AMAZING MAZE V.1.COMAL *
1020 //  *-------------------------*
1030 //  * A.Megens       febr.87  *
1040 //  * for MON/DOS65 systems  *
1050 //  ***************************
1060 //
1070 B:=26
1080 H:=10
1090 DIM D(B,H),B(3),M(2*B+1,2*H+1),V(2*B+1,2*H+1)
1100 C1:=1/(B+H)
1110 C2:=.7
1120 C3:=.8
1130 C4:=.5
1140 E:=B*H
1150 I:=INT(((RND(1)+.5)*B)/2)
1160 T:=1
1170 A$:="400140240124304134324132"
1180 GRAF$:=CHR$(27)+"F"
1190 TEXT$:=CHR$(27)+"G"
1200 FOR X:=0 TO B
1210  FOR Y:=0 TO H
1220   D(X,Y):=0
1230  ENDFOR
1240 ENDFOR
1250 X:=1
1260 Y:=0
1270 D(X,Y):=1
1280 X:=X-1
1290 B:=B-1
1300 H:=H-1
1310 A:=0
1320 P:=0
1330 //  ******** MAIN LOOP MAZE GENERATOR ********
1340 REPEAT
1350  PRINT T,
1360  REPEAT
1370   IF D(X,Y)=0 OR (A+P)=0 THEN
1380    REPEAT
1390     X:=X+1
1400     IF X>B THEN
1410      X:=0
1420      Y:=Y+1
1430      IF Y>H THEN
1440       Y:=0
1450      ENDIF
1460     ENDIF
1470    UNTIL D(X,Y)<>0
1480   ENDIF
1490   A:=0
1500   P:=0
1510   IF X<B THEN
1520    IF D(X+1,Y)=0 THEN
1530     P:=1
1540    ENDIF
1550   ENDIF
1560   IF X>0 THEN
1570    IF D(X-1,Y)=0 THEN
1580     P:=P+2
1590    ENDIF
1600   ENDIF
1610   IF Y<H THEN
1620    IF D(X,Y+1)=0 THEN
1630     P:=P+4
1640    ENDIF
1650   ENDIF
1660   IF Y>0 THEN
1670    IF D(X,Y-1)=0 THEN
1680     A:=1
1690    ENDIF
1700   ENDIF
1710   IF P>0 THEN
1720    A:=A+1
1730    IF P>2 THEN
1740     A:=A+1
1750    ENDIF
1760   ENDIF
1770  UNTIL (A+P)<>0
```



Waddaya mean, user error!?

```
1780  FLAG:=Ø
1790  REPEAT
1800   REPEAT
1810    Q:=3*P+INT(RND(1)*A+1)
1820   UNTIL MID$(A$,Q,1)<>"Ø"
1830   C$:="FUN"+MID$(A$,Q,1)
1840   EXEC: C$,FLAG,X,Y,D(X,Y),C2
1850  UNTIL FLAG<>Ø
1860  PRINT
1870  T:=T+1
1880  IF RND(1)<C1 THEN
1890   X:=INT(RND(1)*B)
1900   Y:=INT(RND(1)*H)
1910  ENDIF
1920 UNTIL T>=E
1930 CLS
1940 D(B-I,H):=D(B-I,H)+4
1950 EXEC: "MAZE"
1960 END.
1970 //
1980 PROC "MAZE"
1990  PRINT GRAF$;
2000  Y:=Ø
2010  FOR X:=Ø TO B
2020   IF X=I THEN
2030    PRINT "Z  ";
2040   ELSE
2050    PRINT "ZXX";
2060   ENDIF
2070  ENDFOR
2080  PRINT "Z"
2090  FOR Y:=Ø TO H
2100   FOR X:=Ø TO B
2110    IN:=D(X,Y)
2120    EXEC: "BINARY",IN,B(1),B(2)
2130    IF B(1)=1 THEN
2140     PRINT "   ";
2150    ELSE
2160     PRINT "Y  ";
2170    ENDIF
2180   ENDFOR
2190   PRINT "Y"
2200   FOR X:=Ø TO B
2210    IN:=D(X,Y)
2220    EXEC: "BINARY",IN,B(1),B(2)
2230    IF B(2)=1 THEN
2240     PRINT "Z  ";
2250    ELSE
2260     PRINT "ZXX";
2270    ENDIF
2280   ENDFOR
2290   PRINT "Z"
2300  ENDFOR
2310  PRINT TEXT$
2320 ENDPROC
2330 //
2340 PROC "FUN1",FLAG,X,Y,D(X,Y),C2
2350  IF RND(1)<C2 THEN
2360   FLAG:=Ø
2370  ELSE
2380   X:=X+1
2390   PRINT "+X";
2400   D(X,Y):=2
2410   FLAG:=1
2420  ENDIF
2430 ENDPROC
2440 //
2450 PROC "FUN2",FLAG,X,Y,D(X,Y),C2
2460  IF RND(1)<(1-C2) THEN
2470   FLAG:=Ø
2480  ELSE
2490   D(X,Y):=D(X,Y)+2
2500   X:=X-1
2510   PRINT "-X";
2520   D(X,Y):=1
2530   FLAG:=1
2540  ENDIF
2550 ENDPROC
2560 //
2570 PROC "FUN3",FLAG,X,Y,D(X,Y),C2
2580  IF RND(1)<C3 THEN
2590   FLAG:=Ø
2600  ELSE
2610   D(X,Y):=D(X,Y)+4
2620   Y:=Y+1
2630   PRINT "+Y";
2640   D(X,Y):=1
2650   FLAG:=1
2660  ENDIF
2670 ENDPROC
2680 //
2690 PROC "FUN4",FLAG,X,Y,D(X,Y),C2
2700  Y:=Y-1
2710  PRINT "-Y";
2720  D(X,Y):=4
2730  FLAG:=1
2740  IF RND(1)<C4 THEN
2750   C2:=1-C2
2760  ENDIF
2770 ENDPROC
2780 //
2790 PROC "BINARY",IN,B(1),B(2)
2800  NUM:=IN
2810  FOR I:=3 TO Ø STEP -1
2820   IF NUM-2^I>=Ø THEN
2830    B(I):=1
2840    NUM:=NUM-2^I
2850   ELSE
2860    B(I):=Ø
2870   ENDIF
2880  ENDFOR
2890 ENDPROC
2900 //
```