

# **RXB**

## **RICH EXTENDED BASIC**

---

Format           RANDOMIZE  
  
                  RANDOMIZE SEED

### Description

The RANDOMIZE command can be found on XB manual page 151 to help explain it's use. RXB unlike any other XB produced has a feature that makes RND and RANDOMZE different and better. RANDOMIZE SEED in RXB is same as TI BASIC randomize seed. Thus in RXB do not expect the same random numbers as you would get with any other XB made. RXB is way more random due to this change different then any other Extended Basic.

### Program

```
Will put hex >3567 into seed | >100 RANDOMIZE
RND example to prove speed   | >110 DIM N(100)
Counter in a FOR loop        | >120 FOR X=1 TO 100
Load Array with random numbers| >130 N(X)=RND
Show that number              | >140 PRINT N(X)
Repeat loop till done         | >150 NEXT X
```

Run this above example in TI BASIC, XB and RXB 2020 to show game type results of program results with new RND

### Options

Random Music programs will sound very very fast due to the speed increase in RXB RND is much faster.

---

Format            RES                            (Uses default values)

                  RES initial line,increment

                  RES initial line,increment,start line-end line

### Description

The RES command is the same as per Extended Basic Manual page 155. The RESEQUENCE command is deleted. The abbreviation RES is the only access name. The RES command now allows a portion of the program to be resequenced. This RES DOES NOT REPLACE any undefined line numbers with 32767. Any undefined line numbers in the program are left as is. This makes it easier to fix if a problem is present. RES cannot be used to move lines from one location to another inside a program. If the new line numbers generated by the RES command would result in a line being moved, a Bad Line Number Error is generated. A Bad Line Number Error is also reported if there are no valid program lines between start line and end line.

### Command

Lines 10 to 50 are renumbered. Line 10 becomes 20, increment is 1.	>RES 20,1,10-50
Lines 700-800 are renumbered. Line 700 becomes 100, increment is 5.	>RES ,5,700-800
Lines 50-80 are renumbered. Line 50 becomes 100, increment is 10. (Default)	>RES ,,50-80
Lines 1000 to last line are renumbered. Line 750 becomes 1000, increment is 10.	>RES 1000,,750-
Lines to 400 are renumbered. First Line becomes 100 (Default), increment is 20.	>RES ,20,-400
Line 40 is renumbered 20.	>RES 20,,40

---

Format           CALL RMOTION(#sprite-number[,...])  
                   CALL RMOTION(ALL[,...])

### Description

The RMOTION subprogram reverses the row-velocity and column-velocity as numbers from -127 to 127. This means that RMOTION simply reverses the direction of the sprite specified so it goes in the opposite direction it was going in. This also means RMOTION ignores 0 and -128, so you can use those to bypass RMOTION if you do not want RMOTION to change the sprite. The fastest and slowest sprite speeds are never affected by RMOTION. This feature adds more power to RMOTION. The ALL feature also allows all sprites on the screen to reverse all at once. ALL may also be called as many times as wanted in a single program line.

### Program

RMOTION reverses the row-velocity and the column-velocity in sprite-number 1.	>100 CALL RMOTION(#1)
This line reverses the motion of all sprites.	>100 CALL RMOTION(ALL)
Line 100 sets up a sprite.	>100 CALL SPRITE(#1,33,2,96,1                     8,99,84)
Line 110 waits for a number higher than .8 randomly.	>110 IF RND<.8 THEN 110
Line 120 reverses the motion of the sprite.	>120 CALL RMOTION(#1)
Continues the program.	>130 GOTO 110

### Options

While characters 144 to 159 are being used, you cannot use sprites.

---

Format

RND

### Description

The RND subprogram in RXB has been replaced with a TI BASIC version as the normal XB RND subprogram is hindered with so much Floating Point as to make it 3 times slower than the TI BASIC version of RND. Extensive testing proves that the new RXB RND is many times faster than the previous version.

There will actually be some programs expecting a particular RND pattern of random numbers that will no longer work the same as a result of this change. But games will appear more random than normal Extended Basic.

The RANDOMIZE seed still works but the results of the that pattern of random numbers will be different than normal XB, thus unless absolutely required will be a bigger benefit than the cost of this XB previous feature.

### Program

RND example to prove speed		>100 DIM N(100)
Counter in a FOR loop		>110 FOR X=1 TO 100
Load Array with random numbers		>120 N(X)=RND
Show that number		>130 PRINT N(X)
Repeat loop till done		>140 NEXT X

Run this above example in TI BASIC, XB and RXB 2015 to show game type results of program results with new RND

### Options

Random Music programs will sound very very fast.

---

Format           CALL ROLLDOWN  
                  CALL ROLLDOWN(repetition)

### Description

ROLLDOWN scrolls screen to the down so repetition will repeat the scroll number of times to down. Repetition can be 1 to 256 max.

### Programs

Roll screen down 2 times	>CALL ROLDOWN(2)
Prints line	>100 PRINT "SCREEN PRINT"
Roll screen down	>110 CALL ROLLDOWN
Repeat the program	>100 GOTO 110
Load X\$ string variable	>100 X\$=" SCROLL DOWN"
Print X\$	>110 PRINT X\$
Roll down 9 times use X\$	>120 CALL ROLLDOWN(9)
Repeat the program	>130 GOTO 100

### Options

New features allow for some special that can take the place of some routines that are slower in XB.

---

Format           CALL ROLLLEFT  
  
                  CALL ROLLLEFT(repetition)

### Description

ROLLLEFT scrolls screen to the left so repetition will repeat the scroll number of times to left.  
Repetition can be 1 to 256 max.

### Programs

Roll screen left 2 times	>CALL ROLLLEFT(2)
Prints line	>100 PRINT "SCREEN PRINT"
Roll screen left	>110 CALL ROLLLEFT
Repeat the program	>120 GOTO 110
Load X\$ string variable	>100 X\$=" SCROLL LEFT"
Print X\$	>110 PRINT X\$
Roll left 9 times use X\$	>120 CALL ROLLLEFT(9)
Repeat the program	>130 GOTO 100

### Options

New features allow for some special that can take the place of some routines that are slower in XB.

---

Format           CALL ROLLRIGHT  
  
                  CALL ROLLRIGHT(repetition)

### Description

ROLLRIGHT scrolls screen to the right so repetition will repeat the scroll number of times to right.  
Repetition can be 1 to 256 max.

### Programs

Roll screen right 2 times	>CALL ROLLRIGHT(2)
Prints line	>100 PRINT "SCREEN PRINT"
Roll screen right	>110 CALL ROLLRIGHT
Repeat the program	>120 GOTO 110
Load X\$ string variable	>100 X\$=" ROLL RIGHT"
Print X\$	>110 PRINT X\$
Scroll right 9 times use X\$	>120 CALL ROLLRIGHT(9)
Repeat the program	>130 GOTO 100

### Options

New features allow for some special that can take the place of some routines that are slower in XB.

---

Format           CALL ROLLUP  
  
                  CALL ROLLUP(repetition)

### Description

ROLLUP scrolls screen to the up so repetition will repeat the scroll number of times to up. Repetition can be 1 to 256 max.

### Programs

Roll screen up 2 times	>CALL ROLLUP(2)
Prints line	>100 PRINT "SCREEN PRINT"
Roll screen UP	>110 CALL ROLLUP
Repeat the program	>120 GOTO 110
Load X\$ string variable	>100 X\$=" SCROLL UP"
Print X\$	>110 PRINT X\$
Roll up 9 times use X\$	>120 CALL ROLLUP(9)
Repeat the program	>130 GOTO 100

### Options

New features allow for some special that can take the place of some routines that are slower in XB.



Example is mixing commands:

```
100 CALL SAMS("ON","MAP",2,237,"OFF")
```

This turns on SAMS read/write Registers, turns on MAP mode, sets 4K page with page 237 than turns off SAMS read/write Registers.

### Programs

This turns on the SAMS mapper.		>110 CALL SAMS("ON")
This reads low half 8K page.		>120 CALL PEEK(16388,L)
This reads high half 8K page.		>130 CALL PEEK(16390,H)
This shows pages used.		>140 PRINT "LOW";L;"HIGH";H
This loads a assembly program.		>150 CALL LOAD("DSK1.CHAR")
This changes low/high 4K pages		>160 CALL SAMS(2,16,3,17)
This loads a assembly program.		>170 CALL LOAD("DSK1.DUMP")
This changes low/high back.		>180 CALL SAMS(2,L,3,H)
This uses a routine in CHAR.		>190 CALL LINK("CHAR")
This changes low/high again.		>200 CALL SAMS(2,16,3,17)
This uses a routine in DUMP.		>210 CALL LINK("DUMP")

The above example program shows one RXB program using two assembly programs with links for both. Thus only 16K of the SAMS was used. 1024K would be 120 assembly support programs. Compatibility of most software assured in RXB AMS support.

### Options:

See ON, OFF, MAP and PASS pages in RXB Documents for more information on SAMS.

## SAMS MAPPER

\*\*\*\*\*

The SAMS card has tons of documents as to its function and use. So to re-explain these docs would be pointless. Read the docs or find some, sorry but the RXB package is already huge.

In PASS mode the mapper register setup is equivalent to:

mapper address		mapper	page num		address range
HEX	Dec		HEX	Dec	memory area
>4004	= 16388	is MR02	= >02	= 02	points to >2000 - >2FFF range
>4006	= 16390	is MR03	= >03	= 03	points to >3000 - >3FFF range
>4014	= 16404	is MR10	= >0A	= 10	points to >A000 - >AFFF range
>4016	= 16406	is MR11	= >0B	= 11	points to >B000 - >BFFF range
>4018	= 16408	is MR12	= >0C	= 12	points to >C000 - >CFFF range
>401A	= 16410	is MR13	= >0D	= 13	points to >D000 - >DFFF range
>401C	= 16412	is MR14	= >0E	= 14	points to >E000 - >EFFF range
>401E	= 16414	is MR15	= >0F	= 15	points to >F000 - >FFFF range

(MR=Mapper Register)

In MAP mode the mapper register setup is equivalent to: EXAMPLE1

mapper address		mapper	page num		address range
HEX	Dec		HEX	Dec	memory area
>4004	= 16388	is MR02	= >10	= 16	points to >2000 - >2FFF range
>4006	= 16390	is MR03	= >11	= 17	points to >3000 - >3FFF range
>4014	= 16404	is MR10	= >12	= 18	points to >A000 - >AFFF range
>4016	= 16406	is MR11	= >13	= 19	points to >B000 - >BFFF range
>4018	= 16408	is MR12	= >14	= 20	points to >C000 - >CFFF range
>401A	= 16410	is MR13	= >15	= 21	points to >D000 - >DFFF range
>401C	= 16412	is MR14	= >16	= 22	points to >E000 - >EFFF range
>401E	= 16414	is MR15	= >17	= 23	points to >F000 - >FFFF range

(MR=Mapper Register)

## SAMS MAPPER

\*\*\*\*\*

In map mode the mapper register setup is equivalent to: EXAMPLE2

mapper address		mapper	page num		address range
HEX	Dec		HEX	Dec	memory area
>4004	= 16388	is MR02	= >62	= 98	points to >2000 - >2FFF range
>4006	= 16390	is MR03	= >63	= 99	points to >3000 - >3FFF range
>4014	= 16404	is MR10	= >64	= 100	points to >A000 - >AFFF range
>4016	= 16406	is MR11	= >65	= 101	points to >B000 - >BFFF range
>4018	= 16408	is MR12	= >66	= 102	points to >C000 - >CFFF range
>401A	= 16410	is MR13	= >67	= 103	points to >D000 - >DFFF range
>401C	= 16412	is MR14	= >68	= 104	points to >E000 - >EFFF range
>401E	= 16414	is MR15	= >69	= 105	points to >F000 - >FFFF range

(MR=Mapper Register)

In MAP mode the mapper register setup is equivalent to: EXAMPLE3

mapper address		mapper	page num		address range
HEX	Dec		HEX	Dec	memory area
>4004	=16388	is MR02	=>1FF9	= 8185	points to >2000 - >2FFF range
>4006	=16390	is MR03	=>1FFA	= 8186	points to >3000 - >3FFF range
>4014	=16404	is MR10	=>1FFB	= 8187	points to >A000 - >AFFF range
>4016	=16406	is MR11	=>1FFC	= 8188	points to >B000 - >BFFF range
>4018	=16408	is MR12	=>1FFD	= 8189	points to >C000 - >CFFF range
>401A	=16410	is MR13	=>1FFE	= 8190	points to >D000 - >DFFF range
>401C	=16412	is MR14	=>1FFF	= 8191	points to >E000 - >EFFF range
>401E	=16414	is MR15	=>2000	= 8192	points to >F000 - >FFFF range

(MR=Mapper Register)

---

Format           SAVE DSK3.PRGM  
                   SAVE DSK2.PRGM,IV254

### Description

The SAVE command functions normally to save XB programs. An additional feature is IV254 may be specified after the SAVE command to convert to Internal Variable 254 format. The IV254 format makes it much more easy to tell an XB program from EA programs when cataloging a disk. Internal Variable files do take up one sector more than XB program format. It should be noted that XB programs smaller than 3 sectors can not be saved in IV254 format.

### Command

Saves to DISK 2 in XB program image format TEST	>SAVE DSK2.TEST
Saves to disk 3 in XB program Internal Variable 254 named STUFF	>sAVE DSK3.STUFF,IV254
Saves to WDS1 in directory EXB XB program Internal Variable 254 named RB	>SAVE WDS1.EXB.RB,IV254

### Options

Allows better cataloging options for saving XB files.

Format                    CALL SCREEN(color-code[,...])  
                           CALL SCREEN("OFF"[,...])  
                           CALL SCREEN("ON"[,...])

Description

See EXTENDED BASIC MANUAL PAGE 165 for more data.  
 RXB has added features of OFF and ON to the SCREEN command. OFF turns off the screen display while the ON turn the screen back on. Use of OFF command allows for writing to screen happens but not visible to user.

Programs

Turn screen to white		>100 CALL SCREEN(16)
Turn off the screen display		>100 CALL SCREEN("OFF")
Prints line but screen off		>110 PRINT "THE SCREEN IS OFF"
Waits for any key		>120 CALL KEY("",5,K,S)
This opens a RS232 port.		>130 CALL SCREEN("ON")
Prints line but screen on		>140 PRINT "NOW SCREEN ON"
Waits for any key		>150 CALL KEY("",5,K,S)
Special effect use of SCREEN		>160 CALL SCREEN(0,2,0,2,0,2)

Options

New features allow for some special effects like draw screen while screen is off and then pop it to user. Or use of the comma to switch colors making some special effects.

---

Format

```
CALL SCROLLDOWN
CALL SCROLLDOWN(repetition)
CALL SCROLLDOWN(repetition,string)
CALL SCROLLDOWN(repetition,string,tab)
```

### Description

SCROLLDOWN scrolls screen to the down so repetition will repeat the scroll number of times down, the string will only display horizontally 32 characters of the string on right side of screen. SCROLLDOWN puts the string on screen and wraps to bottom if string is to long. If the string is empty (null) it will just scroll the screen. Like PRINT TAB will go to next line right of left side of screen each line till end of string. Numbers or variables can be used instead of a string.

### Programs

Scroll down 2 times print PI	>CALL SCROLDOWN(2,PI)
Clear screen for demo	>100 CALL CLEAR
Prints line	>110 PRINT "SCREEN PRINT"
Scroll screen down	>120 CALL SCROLLDOWN
Repeat the program	>130 GOTO 110
Load X\$ string variable	>100 X\$=" SCROLL DOWN"
Print X\$	>110 PRINT X\$
Scroll down 9 times use X\$	>120 CALL SCROLLDOWN(9,X\$)
Repeat the program	>130 GOTO 100

---

Format            CALL SCROLLLEFT

                  CALL SCROLLLEFT(repetition)

                  CALL SCROLLLEFT(repetition,string)

                  CALL SCROLLLEFT(repetition,string,tab)

### Description

SCROLLLEFT scrolls screen to the left so repetition will repeat the scroll number of times left, the string will only display vertically 24 characters of the string on left side of screen. SCROLLLEFT unlike SCROLLRIGHT puts the string on screen and wraps to other side if string is too long. If the string is empty (null) it will just scroll the screen each line till end of string. Numbers or variables can be used instead of a string.

### Programs

Scroll left 2 times print PI	>CALL SCROLLLEFT(2,PI)
Clear screen for demo	>100 CALL CLEAR
Prints line	>110 PRINT "SCREEN PRINT"
Scroll screen left	>120 CALL SCROLLLEFT
Repeat the program	>130 GOTO 110
Load X\$ string variable	>100 X\$=" SCROLL LEFT"
Print X\$	>110 PRINT X\$
Scroll left 9 spaces use X\$	>120 CALL SCROLLLEFT(9,X\$)
Repeat the program	>130 GOTO 100

---

Format

```
CALL SCROLLRIGHT
CALL SCROLLRIGHT(repetition)
CALLL SCROLLRIGHT(repetition,string)
CALL SCROLLRIGHT(repetition,string,tab)
```

### Description

SCROLLRIGHT scrolls screen to the right so repetition will repeat the scroll number of times right, the string will only display vertically 24 characters of the string. SCROLLRIGHT unlike SCROLLLEFT puts the string on screen and does not wrap to other side if string is too long. If the string is empty (null) it will just scroll the screen each line till end of string. Numbers or variables can be used instead of a string.

### Programs

Scrollright 2 times print PI		>CALL SCROLLRIGHT(2,PI)
Clear screen for demo		>100 CALL CLEAR
Prints line but screen off		>110 PRINT "SCREEN PRINT"
Scroll screen right		>120 CALL SCROLLRIGHT
Repeat the program		>130 GOTO 110
Load X\$ string variable		>100 X\$=" SCROLL RIGHT"
Print X\$		>110 PRINT X\$
Scroll right 9 spaces use X\$		>120 CALL SCROLLRIGHT(9,X\$)
Repeat the program		>130 GOTO 100

---

Format

```
CALL SCROLLUP
CALL SCROLLUP(repetition)
CALL SCROLLUP(repetition,string)
CALL SCROLLUP(repetition,string,tab)
```

### Description

SCROLLUP scrolls screen up so repetition will repeat the scroll number of times to up, the string will only display horizontally 32 characters of the string. SCROLLUP puts the string on screen and wraps to top if string is too long. If the string is empty (null) it will just scroll the screen each line till end of string. Numbers or variables can be used instead of a string.

### Programs

Scroll up 2 times print PI	>CALL SCROLLUP(2,PI)
Clear screen for demo	>100 CALL CLEAR
Prints line but screen off	>110 PRINT "SCREEN PRINT"
Scroll screen UP	>120 CALL SCROLLUP
Repeat the program	>130 GOTO 110
Load X\$ string variable	>100 X\$=" SCROLL UP"
Print X\$	>110 PRINT X\$
Scroll up 9 spaces use X\$	>120 CALL SCROLLUP(9,X\$)
Repeat the program	>130 GOTO 100

SIZE                    command or subprogram                    PAGE S11

---

Format                SIZE  
                      CALL SIZE

### Description

See EXTENDED BASIC MANUAL PAGE 169 for more data.  
RXB has added many more features to SIZE. RXB shows the size and memory address of VDP, RAM and SAMS. Very useful for XB or Assembly programmers. EXAMPLE:

```
>SIZE
 11840 Bytes of Stack Free
 24488 Bytes of Program
 8192  Bytes of Assembly
 * PAGE NUMBER = LOCATION *
 2     Page = >2000 - >2FFF
 3     Page = >3000 - >3FFF
 10    Page = >A000 - >AFFF

 11    Page = >B000 - >BFFF
 12    Page = >C000 - >CFFF
 13    Page = >D000 - >DFFF
 14    Page = >E000 - >EFFF
 15    Page = >F000 - >FFFF
 * MEMORY UNUSED and FREE *
>37D7 VDP Free Address
>0958 VDP STACK Address
>FFE7 Program Free Address
>A040 Program End Address
>2000 RAM Free Address
>4000 RAM End Address
```

This shows normal XB values but also includes more useful things like Assembly free and SAMS pages used and where these pages are. Lastly it shows VDP STACK location, First free VDP address, XB RAM First free address and End address. Lastly first free Assembly address and End address used. SAMS size is not reported just like Foppy size or hard drive is'nt!

Format

SIZE

CALL SIZE

## Command

May only be used from command mode. | >SIZE

## Programs

May only be used from program mode. | >100 CALL SIZE

Delay for keypress. | >110 CALL KEY("",0,K,S)

Set up for Assembly support. | >120 CALL INIT

Shows memory used including Assembly space free. | >130 CALL SIZE

Set VDP STACK to >1820 hex. | >140 CALL VDPSTACK(6176)

Show VDP STACK location. | >150 CALL SIZE

Delay for keypress. | >150 CALL KEY("",0,S,S)

Set XB RAM to >A000 hex. | >160 CALL PRAM(-24576)

Shows 64 more bytes of XB RAM for use in XB. | >170 CALL SIZE

Format           CALL MOTION(STOP[,...])

#### Description

The STOP command is a option in the MOTION subprogram. STOP does exactly what you would expect, stop all sprite motion and freezes the sprites in place.

#### Programs

See MOTION subprogram for examples of use of STOP.

---

Format           CALL SWAPCHAR(character-code,character-code  
                 [,...])

### Description

The SWAPCHAR subprogram switches the first character-code character definition with the second character-code character definition. That means they swap definitions. The characters range from 30 to 159.

### Programs

Line 100 swaps character-code 65 with character-code 97.		>100 CALL SWAPCHAR(65,97)
Line 100 defines character-code 128 and character-code 159.		>100 CALL CHAR(128,"F0F0F0F0F0F0F0F0",159,"0F0F0F0F0F0F0F0F0")
Line 110 swaps them, then will swap space with character 128		>110 CALL SWAPCHAR(128,159,32,128)
Line 120 continues program.		>120 GOTO 110
Try this one on for weird.		>100 CALL SWAPCHAR(31,32,31,32)
		>110 CALL INVERSE(31)
		>120 GOTO 100

---

Format           CALL SWAPCOLOR(character-set,character-set  
                  [,...])

                  CALL SWAPCOLOR(#sprite-number,#sprite-number  
                  [,...])

### Description

The SWAPCOLOR subprogram swaps foreground and background colors of the first set with the second set. Or swaps the first sprite-number color with the second sprite-number color. The character-set numbers are given below:

	set-number		character-codes
	~~~~~		~~~~~
	0	-----	30 to 31
	1	-----	32 to 39
	2	-----	40 to 47
	3	-----	48 to 55
	4	-----	56 to 63
	5	-----	64 to 71
	6	-----	72 to 79
	7	-----	80 to 87
	8	-----	88 to 95
	9	-----	96 to 103
	10	-----	104 to 111
	11	-----	112 to 119
	12	-----	120 to 127
	13	-----	128 to 135
	14	-----	136 to 143
(also sprite table)	15	-----	144 to 151
(also sprite table)	16	-----	152 to 159

---

Format           CALL SWAPCOLOR(character-set,character-set  
                  [,...])

                  CALL SWAPCOLOR(#sprite-number,#sprite-number  
                  [,...])

### Programs

Swap foreground and background colors of set 15 with set 5.		>100 CALL SWAPCOLOR(15,5)
Line 100 sets up two sprites on screen.		>100 CALL SPRITE(#1,65,2,99,9 9,9,9,#2,66,16,88,88,22,33)
Line 110 swaps sprite #1 color with sprite #2 color.		>110 CALL SWAPCOLOR(#1,#2)
Continue program.		>120 GOTO 110

Format                      CALL USER(quoted-string)  
                                    CALL USER(string-variable)

### Description

The USER subprogram overrides the normal editor of edit mode of XB and reads a DV80 file into the key scan routine as if the user was keying it in.

That means Batch Processing is creating XB programs from DV80 files, Editing XB programs, MERGING, Saving, and RUNNING XB programs. Also RESequencing, adding lines, or deleting lines, and re-writing lines from the DV80 file.

Every line to be input from the DV80 file MUST END WITH A CARRIAGE RETURN! A line of input may be up to 588 characters in length. The editor will error out if the crunch buffer is full, reporting a \*Line Too Long\* error. (Over 163 tokens)

Other errors will be reported but will not stop the process of USER continuing to input lines. To find errors in the DV80 file the input lines are shown on screen as they are input into the editor, and errors will be reported. So you must observe the screen for errors to test the DV80 file.

USER will stop after reaching the end of the file. But USER can have its operation suspended CALL POKEV(2242,0) will halt USER and CALL POKEV(2242,9) will resume USER.

INPUT and ACCEPT will try to read from USER if it is not turned off. On the other hand DV80 files can go directly into a INPUT or ACCEPT prompts. Turn off USER to be safe though.

USER will only report errors upon opening, thus if incorrect device or filename then USER reports \* USER ERROR \* and just closes the USER file, thus ending operation of USER.

Example files are included with RXB to show and explain the use of USER. The batch processing USER subprogram opens a new world to the RXB programmer.

Additionally new commands like CALL VDPSTACK and CALL PRAM used with CALL USER means you can modify the entire XB memory in both VDP and RAM from a BATCH file.

Possibilities are almost endless!

---

### Programs

This line starts USER to use Batch processing on a file called FILENAME	>CALL USER("DSK1.FILENAME")
Line 100 is same as above. but within a program.	>100 CALL USER("DSK1.FILE")
Line 100 variable A\$ equals a String-variable path name. Line 110 starts USER to use Batch processing on A\$	>100 A\$="DSK.VOLUME.FILE"   >110 CALL USER(A\$)
Save this program as LOAD.	>100 CALL USER("DSK1.BATCH")

Here is an example DV80 file you save with the name BATCH.

```
! BATCH file for using
NEW and CALL FILES and RUN. cr
cr
CALL XB("DSK1.A-PROGRAM",#) cr
! The # is 0 to 15 (see FILES)
```

The above DV80 file uses cr to mean Carriage Return. And # is for the number of files you wish open. A-PROGRAM is the name of the XB program that needs a certain number of files open.

### Options

To many to list out. See BATCH for demo.

---

Format           CALL VCHAR(row,column,character-code)

                  CALL VCHAR(row,column,character-code,  
                  repetition[,...])

### Description

See EXTENDED BASIC MANUAL page 188 for more data. The only syntax change to VCHAR is the auto-repeat function. Notice the new auto-repeat must have the repetitions used or it gets row confused with repetitions. Also RXB HCHAR is now in ROM.

### Programs

This line puts character 38 at row 1 column 1 for 99 times, then puts character code 87 at row 9 column 1	>100 CALL VCHAR(1,1,38,99,9,1 ,87)
Fills screen with characters.	>100 CALL VCHAR(1,1,32,768,1, 1,65,768,1,1,97,768,1,1,30, 768) :: GOTO 100

### Options

CALL VCHAR is now written in Assembly so much faster is faster than normal XB, also as separate line numbers are needed to continue placing characters on screen. See HCHAR, HPUT, VPUT, HGET and VGET.

Format           CALL VDPSTACK(numeric-variable)

#### Description

The VDPSTACK subprogram allows change of location of the VDP STACK in VDP RAM. Care must be taken to where you place the stack after all any over write or change can crash XB. Normal VDP stack location is 2392 in decimal >0958 in Hex. Some XB programs like The Missing Link use 6176 or >1820 Hex. Another location would be like 4096 which is >1000 in Hex. Combine PRAM with VDPSTACK and Assemblby can be loaded into any memory locations previously very hard to use. That required special loaders so now RXB has PLOAD and PSAVE to get around these problems of loading anywhere in 32K now.

#### Programs

This line clears screen.	>100 CALL CLEAR
Set VDP STACK location.	>110 CALL VDPSTACK(6176)
Display it.	>120 PRINT ">1820 STACK LOCAT ION"
Show results.	>130 CALL SIZE
Wait for key pressed.	>140 CALL KEY("",0,S,S)
Set VDP STACK location.	>150 CALL VDPSTACK(4096)
Display it.	>160 PRINT ">1000 STACK LOCAT ION"
Display it.	>170 CALL SIZE

#### Options

See PRAM for similar change to RAM locations. Also see PLOAD and PSAVE for loading anywhere in 32K RAM.



---

Format           CALL VGET(row,column,length,string-variable  
                 [,...])

#### Description

The VGET subprogram returns into a string-variable from the screen at row and column. Length determines how many characters to put into the string-variable. Row numbers from 1 to 24 and column numbers from 1 to 32. Length may number from 1 to 255. If VGET comes to the bottom of the screen then it wraps to the top of screen.

#### Programs

The program to the right will get into string-variable E\$ the 11 characters at row 5 and column 9.

```
>100 CALL VGET(5,9,11,E$)
```

The program to the right will get into string-variable M\$ the 5 characters at row 1 and column 3, then put into string-variable Q\$ the 1 character at row 9 and column 3, then put into string-variable N\$ the 32 characters at row 24 and column 1.

```
>100 CALL VGET(1,3,5,M$,9,3,1
,Q$,24,1,32,N$)
```

#### Options:

See HPUT, VPUT, and HGET.

---

Format           CALL VPUT(row,column,string[,...])  
                   CALL VPUT(row,column,string-variable[,...])

### Description

The VPUT subprogram puts a string or string-variable or number or number variable or constant onto the screen at row and column. The row numbers from 1 to 24 and column numbers from 1 to 32. If the string or number or numeric variable or string-variable or constant being put onto screen goes to an bottom it wraps to the top screen just like VCHAR does. VPUT runs from ROM.

### Programs

Line 100 puts string "THIS" on the screen at row 10 and column 4.		>100 CALL VPUT(10,4,"THIS")
Line 110 sets string-variable A\$ equal to string "VPUT"		>110 A\$="VPUT"
Line 120 puts string "is" at row 11 and column 5, then puts string-variable A\$ at row 10 and column 6.		>120 CALL VPUT(11,5,"is",10,6 ,A\$)
Puts 456 at row 10 col 15		>100 CALL VPUT(10,15,456)

### Options:

CALL VPUT is now written in Assembly so much faster is faster than normal then XB DISPLAY AT(row,column)

(But a vertical version.)

See HCHAR, VCHAR, HPUT, HGET and VGET.

---

Format           CALL VPUT(row,column,string[,...])  
                   CALL VPUT(row,column,string-variable[,...])

### Description

The VPUT subprogram puts a string or string-variable or number or number variable or constant onto the screen at row and column. The row numbers from 1 to 24 and column numbers from 1 to 32. If the string or number or numeric variable or string-variable or constant being put onto screen goes to an bottom it wraps to the top screen just like VCHAR does. VPUT runs from ROM.

### Programs

Line 100 puts string "THIS" on the screen at row 10 and column 4.	>100 CALL VPUT(10,4,"THIS")
Line 110 sets string-variable A\$ equal to string "VPUT"	>110 A\$="VPUT"
Line 120 puts string "is" at row 11 and column 5, then puts string-variable A\$ at row 10 and column 6.	>120 CALL VPUT(11,5,"is",10,6 ,A\$)
Puts 456 at row 10 col 15	>100 CALL VPUT(10,15,456)

### Options:

CALL VPUT is now written in Assembly so much faster is faster than normal then XB DISPLAY AT(row,column)

(But a vertical version.)

See HCHAR, VCHAR, HPUT, HGET and VGET.



This is a copy of the RXB title screen:

```
*****  
* VERSION = 2022 *  
*****  
*   R X B   *  
*           *  
*   creator *  
*           *  
* Rich Gilbertson *  
*****
```

>> press ===== result <<

ANY KEY = DSK#.LOAD

ENTER = DSK#.UTIL1

(COMMA) , = DSK#.BATCH

SPACE BAR = RXB COMMAND MODE

(PERIOD) . = EDITOR ASSEMBLER

NOTE: 0 (ZERO) defaults to WDS1.LOAD or after pressing

ENTER defaults to WDS1.UTIL1

This is a explanation of the keys of the MENU screen:

-----  
(any key) = DSK#.LOAD

While the screen shows menu RXB is selected pressing any key will be the drive that DSK#.LOAD will be run from. RAMDISK number keys 1 to 9 or the alpha keys A to z. Pressing 0 (zero) key will run WDS1.LOAD

-----  
(ENTER key) = DSK#.UTIL1

While the screen shows menu RXB is selected pressing ENTER key allows Assembly Programs to be used. Pressing any key will be the drive that DSK#.UTIL1 will be run from. RAMDISK number keys 1 to 9 or the alpha keys A to z. Pressing 0 (zero) key will run WDS1.UTIL1

-----  
(COMMA) , = DSK#.BATCH

While the screen shows menu RXB is selected pressing COMMA key runs DSK#.BATCH DSK#.BATCH defaults to DSK1 if BATCH not found will default to command mode. For more information on this feature read USER in the RXB information on BATCH FILE SYSTEM below.

-----  
(SPACE BAR) = RXB COMMAND MODE

Pressing the SPACE BAR results in XB command mode. (Same as waiting a few seconds just like normal XB does.)

-----  
(PERIOD) . = EDITOR ASSEMBLER

Pressing the . (PERIOD) key will switch to EDITOR ASSEMBLER menu. Pressing the .

-----  
(ZERO) 0 = WSD1.LOAD

Pressing the 0 (ZERO) key will start a WSD1.LOAD to execute from hard drive 1. If the root directory has a LOAD program.

## BATCH FILE SYSTEM:

-----  
 CALL USER overrides the normal edit mode by allowing a DV80 file to take control. This allows conversions from DV80 to XB program or DV80 to XB MERGE format or loading files, re-sequencing them, and saving or merging or adding lines through another DV80 file. All variables used through CALL USER are not affected so from a running program more lines or variables can be added to the size of the program without losing anything. Of course the RUN command will as always clear all variables before the program is run, this feature can be turned off with a CALL LOAD. (PRESCAN OFF)

As the USER subprogram can override the Editor many features can be bypassed. Example:

```

NEW                                cr
OLD DSK1.XBPROGRAM                 cr
RES 11,3                            cr
MERGE "DSK1.MERGEPM"               cr
SIZE                                cr
SAVE "DSK1.NEWPROGRAM"              cr
RUN                                  cr
NEW                                  cr
OLD DSK1.LOAD                       cr

```

The above is a good example of a DV80 Batch file for RXB. Note that there must be a CHR\$(13) or Carriage Return after every input line. If not then RXB assumes the it is the same line. But even that is not much of a problem as RXB allows 21 lines of input per program line. You can make them even longer if you want in USER.

## INPUT/OUTPUT ACCESS:

-----  
 CALL IO controls the 9901 CRU chip. Sound lists can be played independently of current status. (i.e. type in a program while playing music from VDP/GROM.) Control Register Unit can turn on/off single bits of CRU address bus. (i.e. cards/chips) Cassette direct bus control. (i.e. no menu input/output, verify)

## REDO KEY RESTORED (Was removed in RXB2001 to RXB2012):

-----  
 The REDO (FCTN 8) is RESTORED in RXB2015. USER needed a buffer that would not be molested or modified by CALL LINK, CALL LOAD or routines that need a buffer and usually use the same area that USER previously used. So to update and eliminate questions of compatibility the USER buffer was installed in place of the Edit recall buffer (REDO). The REDO key was not considered to be of much use anyway as the Crunch Buffer is 163 tokens long and in non-tokenized form the Edit recall buffer is only 152 bytes long. That is why when REDO is pressed only part of the line last typed in was recalled to screen. Additionally COPY lines, and MOVE lines commands can do the same thing as REDO could, so not much of anything is lost because it is assumed a TEXT EDITOR will be used to create programs in RXB then use CALL USER.

## PROGRAM DEVICE NAMES ACCESS:

-----  
 New access names established as devices are now available. By using any TRUE DSR (Device Service Routine) you may now access the Editor Assembler main menu by typing 'EA' within Basic or RXB. Example: RUN "EA" or OLD EA or DELETE "EA"  
 You may also access RXB from Editor Assembler or Basic or even another cartridge. Example: OLD XB or DELETE "XB" from Basic.  
 At any Editor Assembler device prompt type 'XB' then enter.

## FOR ASSEMBLY LANGUAGE PROGRAMMERS:

-----  
 CALL MOVES is a new command that is a GPL command converted and added to RXB to give total control over every type of memory with in the TI-99/4A. MOVES works with address or strings to copy, over-write or move blocks of memory of any type of memory. RAM, VDP, GROM, GRAM, and ROM can be accessed by CALL MOVES.

#### RXB TO ASSEMBLY DIRECT ACCESS BY ADDRESS:

-----

EXECUTE is much faster than the traditional LINK routine built into XB. The main problem with LINK is it checks everything and pushes everything onto the VDP stack. After getting to Assembly it pops everything off the stack for use or pushes what is to be passed to XB onto the stack. EXECUTE on the other hand just passes a address to a 12 byte Assembly program in Fast RAM and RTWP ends the users program. A LINK will use up 6 bytes for the name, 2 bytes for the address and wastes time checking things.

The advantage to EXECUTE is you use LOAD or MOVE or MOVES to place the values needed directly into the registers then do it. EXECUTE uses less space, is faster, and is easy to debug.

#### SAMS SUPPORT ROUTINES:

-----

The SAMS has support routines built into RXB. CALL SAMS("MAP") will turn the SAMS mapper on. CALL AMS("PASS") turns SAMS mapper to pass mode. CALL SAMS("ON") will turn on the read/write lines of the mapper. CALL SAMS("OFF") turns off the read/write lines. With these commands pages of memory can be written with a CALL LOAD or read with a CALL PEEK.

RXB AMS SUPPORT USES NO ASSEMBLY OR CALL LINKS. That means up to 1 meg of 4K pages in entire 32K from RXB. That is impossible to do from XB as you have to load your normal support somewhere in 32K of assembly for everyone else not using RXB.

GPL is where all the support routines are stored in RXB so not one byte is wasted on assembly support. That also means not one byte of SAMS memory is wasted on control routines.

Speaking of control CALL SAMS switches 4K pages in the 32K SAMS. CALL SAMS uses boundry symbols upper case only.

i.e. 2 = >2000, 3 = >3000, A = >A000, B = >B000, C = >C000,  
D = >D000, E = >E000 and F = >F000

#### RND FUNCTION REPLACED

-----

Extended Basic RND has been replaced with the TI BASIC RND as the normal XB version of RND was hindered by to much Floating Point that is very slow for use just to get a random number. Also the XB RND was insanely complicated and bloated.

INTERRUPT SERVICE ROUTINE CONTROL (ISROFF and ISRON)

-----

ISR (Interrupt Service Routine) like MOUSE or Screen dumps or any special program like XB Packer use a ISR. The problem with these programs is unless they are written to work with new devices, a lock-up occurs. EXAMPLE: running a mouse routine and XB Packer. They were never made to work together. RXB now has a handle on this. CALL ISROFF turns off the interrupt and saves the address for turning it back on. CALL ISRON restarts the interrupt. As several pages of the AMS can be used with interrupts a whole new world of programming is now possible.  
 NO ASSEMBLY IS USED OR CALL LINKS. Absolute compatibility.

4K PROGRAM IMAGE FILE LOADER AND SAVER (PLOAD and PSAVE)

-----

Hidden loaders were created to overcome the slow loading speed of CALL LOAD. The disadvantage of a hidden loader is it can only load one assembly support program at a time. PLOAD loads program image files of 4K, and PLOAD can load as many times as needed within one RXB program. PSAVE is the opposite and creates the program image files of the 4K anywhere in memory. Lastly loading 200K into the SAMS card is easy with PLOAD. A simple loop can load each SAMS 4K page with PLOAD. Each address boundry is in PSAVE or PLOAD like SAMS uses boundry symbols upper case only. i.e. 2 = >2000, 3 = >3000, A = >A000, B = >B000, C = >C000, D = >D000, E = >E000 and F = >F000

SAVE FILES IN INTERNAL VARIABLE 254 OR PROGRAM IMAGE FORMAT

-----

RXB allows XB programs to load or be saved in two formats as previously, but now RXB allows more control of this feature. Normally XB will save files in Program Image format if these programs are small enough to fit in VDP memory. If these XB programs are larger then what will fit in VDP then XB programs will be saved in Internal Variable 254 format. RXB has a added feature added to save command. IV254 is the new feature.  
 EXAMPLE: SAVE DSK3.TEST,IV254

JOYSTICK and SPRITE MOTION CONTROL with KEY built FIRE button  
 -----

As normal XB JOYSTICK and SPRITE controls were separate commands this slowed down response in XB games and utilities. The main issue was these commands were not combined. RXB added two new commands to the arsenal but also added CALL KEY and also added a IF THEN into the mix. Thus CALL JOYMOTION acts just like CALL JOYST + CALL KEY + CALL MOTION + IF FIRE THEN line number To bring even more to the table is an INDEX value for SPRITES.  
 EXAMPLE:

```
CALL JOYMOTION(key-unit,x-return,y-return,#sprite,
row-index,column-index,key-return-variable) GOTO line-number
```

key-unit,x-return,y-return are like normal XB JOYST  
 #sprite,row-index,column-index are like XB MOTION but dot based  
 key-return-variable is just like XB KEY key variable  
 GOTO line-number is like XB IF KEY THEN line-number

The GOTO is not required nor is the key-return-variable as these are optional depending on your needs.

JOYSTICK and SPRITE LOCATE CONTROL with KEY built in FIRE button  
 -----

As normal XB JOYSTICK and SPRITE controls were separate commands this slowed down response in XB games and utilities. The main issue was these commands were not combined. RXB added two new commands to the arsenal but also added CALL KEY and also added a IF THEN into the mix. Thus CALL JOYLOCATE acts just like CALL JOYST + CALL KEY + CALL MOTION + IF FIRE THEN line number  
 EXAMPLE:

```
CALL JOYLOCATE(key-unit,x-return,y-return,row-index,column-index,
#sprite,dot-row,dot-column),key-return-variable) GOTO line-number
```

key-unit,x-return,y-return are like normal XB JOYST  
 #sprite,row-index,column-index are like XB LOCATE but dot based  
 key-return-variable is just like XB KEY key variable  
 GOTO line-number is like XB IF KEY THEN line-number

The GOTO is not required nor is the key-return-variable as these are optional depending on your needs.

## RAM MEMORY MANAGER (CALL PRAM)

-----

New way to use RXB way ahead of any other XB made is PRAM that allows you to change the size of RAM in upper 24K of RAM. Normally >A040 is the end of RAM in XB as it starts going from high RAM >FFFC down to lowest toward >A040 this allows 64 bytes not used but was for the TI Debugger to use. The PRAM command changes the location of the end of XB RAM. Normally XB RAM is >A040 in hex so the PRAM command allows changing this location to as low as 298 bytes of XB RAM. Any location from >A000 to >FEBE is a valid change in PRAM. Thus -322 decimal or >FEBE hex is highest address is -25576 decimal or >A000 hex lowest address. That tops our XB RAM to 64 more bytes then normal at max or down to 298 bytes of RAM. How come no one else thought of this? (Need to fix program start)

## VDP STACK MEMORY MANAGER (CALL VDPSTACK)

-----

Normal VDP stack location is 2392 in decimal >0958 in Hex. Some XB programs like The Missing Link use 6176 or >1820 Hex. Another location would be like 4096 which is >1000 in Hex. The VDPSTACK subprogram allows change of location of the VDP STACK in VDP RAM. Care must be taken to where you place the stack after all any over write or change can crash XB. Changing the VDP stack location allows changes in type of VDP mode being used like TEXT mode or Multi colored mode.

## FILES BUFFEER MEMORY MANAGER (CALL FILES)

-----

The FILES subprogram differs from the Disk Controller FILES on the CorComp, TI, Myarc or Parcom versions. All of these require a NEW after CALL FILES. NEW is executed after the FILES subprogram in RXB, no need to use NEW it is built into FILES. Also RXB FILES accepts values from 0 to 15 unlike the other FILES routines that can only accept 1 to 9. Each open file reduces VDP by 534 bytes, plus each file opened will use 518 bytes more. CALL FILES(0) will display 5624 Bytes of Stack free and 24488 Bytes of Program space free. At this point up to 15 files may be open at the same time. Not recommended but possible. Thus RXB 0 files now is possible in RXB or up to 15.

## SIZE REPORT CHANGE

-----  
 RXB has a major change to SIZE routine not just adding CALL SIZE but the report itself is extensively more useful.

>SIZE press enter

Screen advances and you see:

```

>SIZE
11840 Bytes of Stack Free
24488 Bytes of Program Free
8192 Bytes of Assembly Free
256 Pages 1024 K SAMS
2   Page = Address >2000
3   Page = Address >3000
10  Page = Address >A000
11  Page = Address >B000
12  Page = Address >C000
13  Page = Address >D000
14  Page = Address >E000
15  Page = Address >F000
>37D7 VDP Free Address
>0958 VDP STACK Address
>FFE7 Program Free Address
>A040 Program End Address
>2000 RAM Free Address
>4000 RAM End Address

```

>cursor flashing

As you can see much more information than you are used to seeing about memory of XB and system. Note first off the display of Assembly Free memory and if you have a SAMS. If you have a SAMS you also see the pages used and at the address in Hex where it resides. Next is address of first free VDP Address and below that you VDP Stack location. For XB itself you also see the XB program first free address and the End Address for XB program space. Lastly the first free RAM in Assembly lower 8k and last address used by Assembly.

## RXB FIXES TO XB REQUESTED BY USERS

-----

RXB has numerous fixes thru the years a few will be mentioned here as far back as 1983 when I bought my TI99/4A.

Recently asked to fix RANDOMIZE SEED not working with the CALL LINK in XB, so I added a line to reset RANDOM SEED upon use of the CALL LINK. Your welcome.

RXB and XB had a issue with PRINT that worked fine in BASIC and a fix was made to solve this very rare issue. You might have seen it when edge characters were improperly shown.

CALL FILES(0) never worked in BASIC or XB but does work in RXB now. This meant a update to SIZE routine too.

Another XB bug was this example:

```
10 PRINT
LIST
ACCEPT A
```

Now a error is produced unlike version 110 XB crashes.  
RXB shows this instead \* Only legal in a program \*

## THANKS TO LEE STEWART

-----

RXB 2022 has muliple routines now in Assembly to speed up these routines ALPHALOCK, CLEAR, CLEARPRINT, CHARSET, COLLIDE, INIT, HCHAR, HEX, ISROFF, ISRON, VCHAR, HPUT, HGET, ROLLDOWN, ROLLLEFT, ROLLDOWN, ROLLUP, SCROLLDOWN, SCROLLLEFT, SCROLLRIGHT, SCROLLUP, VPUT, VGET and the character loader all are Assembly in ROM. Expect next version of RXB to have even more Assembly for former GPL routines thanks to help from Lee Stewart. Specifically CLEAR, HPUT, VPUT, HCHAR, VCHAR, SCROLL, ROLL and CLEARPRINT are all speedy due to be Assembly now instead of GPL.

CALL CHAR(ALL,pattern-identifier[,...])

CALL CHARSET(ALL)

CALL COLOR(ALL,foreground-color,background-color[,...])

CALL DISTANCE(#sprite,#sprite,numeric-variable[,...])

CALL FILES(number) {0 to 15 can be used now}

CALL GCHAR(row,column,numeric-variable[,...])

CALL HCHAR(row,column,character-code,repitition[,...])

CALL JOYST(key-unit,x-return,y-return[,...])

CALL KEY(key-unit,return-variable,status-variable[,...])

CALL KEY(string,key-unit,return-variable,status-variable[,...])

CALL MAGNIFY(magnification-factor[,...])

CALL MOTION(ALL,row-velocity,column-velocity[,...])

CALL MOTION(GO[,...])

CALL MOTION(STOP[,...])

CALL SCREEN(color[,...])

CALL SCREEN(ON[,...])

CALL SCREEN(OFF[,...])

CALL VCHAR(row,column,character-code,repitition[,...])



TEXAS INSTRUMENTS  
HOME COMPUTER

PRESS

- 1 FOR TI BASIC
- 2 FOR RXB 2022G
- 3 FOR REA 2022G

version = 2021

R X B

creator

RICH GILBERTSON

>> Press ===== result <<

ANY KEY = DSK#.LOAD

ENTER = DSK#.UTIL1

(COMMA) , = DSK#.BATCH

SPACE BAR = RXB COMMAND MODE

(PERIOD) . = EDITOR ASSEMBLER



## R I C H E D I T O R &amp; A S S E M B L E R V = 2 0 2 1

S        S E T P A T H N A M E S

D        D I R E C T O R Y

A        A S S E M B L E R

E        E D I T O R

X        X B P R O G R A M

L        L O A D A N D R U N

P        P R O G R A M F I L E

.        R X B

```
*****  
* RXB Editor Assembler Version 2015 *  
*****
```

REA is a new completely re-written Editor Assembler module.  
Any code not needed was removed, and this left room for many  
new features. TI BASIC support has been removed to add in the  
features like catalog a drive and set pathnames.

This is a copy of the REA title screen:

```
Rich Editor & Assembler V=2015  
-----  
  
S SET PATHS NAMES  
  
D DIRECTORY  
  
A ASSEMBLER  
E EDITOR  
X XB PROGRAM  
L LOAD and RUN  
P PROGRAM FILE  
  
. R X B
```

This is a copy of the REA Configure Paths:

\* CONFIGURE PATHS \*

1 DSK1.EDIT1

2 DSK1.ASSM1

3 DSK1.SOURCE

4 DSK1.OBJECT

5 DSK1.LIST

6 OPTIONS: L

CTRL 1 - 5 DRIVE SELECTION

ANY OTHER KEY TO MAIN MENU

#### S SET PATH NAMES

Sets path of Editor, Assembler, source, object, and list files. Selection of 1 to 6 allows a input like as in previous Editor Assembler version including REA. Selection of CTRL 1 to 5 will allow single selection of drive number for that path. As an example is select CTRL 1 and the number 1 in path DSK1.EDIT1 will beep and ask for a drive number or letter. Another beep indicates selection made and shows the change.

#### E EDITOR

Has a arrow to indicate which option has been selected, thus the user will no longer make a mistake of saving a blank file over the original that he actually meant to load or save. Also as Edit path is preset the loading is automatic for the Editor and the file to load. Save file still asks for a path name and file. Print also asks for device or path name.

i.e. DSK.VOLUMENAME.EDIT1 or WDS1.DIRECTORY.SUBDIRECTORY.EDIT1

The directory will load the selected file if this option is used. See Directory for features.

#### A ASSEMBLER

Assembler has no menu selection as CONFIGURE PATHS does this. The ASSM1 path from S SET PATH automatically loads Source, Object, List file paths and Options. A Assembler key press from main menu starts the Assembler, but SET PATH must be first.

i.e. DSK.VOLUMENAME.ASSM1 or WDS1.DIRECTORY.SUBDIRECTORY.ASSM1

The directory will replace the selected file if this option is used. See Directory for features.

#### L LOAD and RUN

The directory will load the selected file if this option is used. After loading a file all the link names will be displayed including all support routines. Using arrow keys the selected link name can be executed by pressing ENTER key. Up to 80 link names will be displayed on screen thus arrow keys to select a program name to run. See Directory for features.

#### P PROGRAM FILE

By pressing a single key then enter, DSK#.UTIL1 is displayed and executed. # indicates the key pressed A to Z or 1 to 9. Pressing 0 (zero) runs WDS1.UTIL1 at PROGRAM FILE. The directory will load the selected file if this option is used. The lower 8K support routines normally only loaded by the EA3 option are now loaded by this option too. So users can load FORTH, FORTRAM, and C programs from the EA5 prompt.

#### X XB PROGRAM

New feature that prompts for a XB program file to run. If the file or device errors out, then a return to RXB command mode is done. The \* R X B \* and a flashing cursor indicates the XB command mode. By pressing a single key then enter, DSK#.LOAD is displayed and executed. # indicates the key pressed. The directory will load the selected file if this option is used. See Directory for new features.

## D DIRECTORY

A new feature that prompts for a device name. EXAMPLE: DSK1. The period MUST be included if the full device name is used. Or type in a path name EXAMPLE: WDS1.DIRECTORY. The quicker way is to just type a number or letter then enter. Thus DSK#. is used and the key pressed represents the # used. While the catalog is being scrolled on screen, ANY KEY will pause the display and reading of a disk, an arrow will appear next to the file read and the ARROW KEYS will move the arrow up or down. (FCNT/CTRL optional). To page forward or backward a screen at a time press left and right arrow keys. The arrow last pointing to will stay at the top or bottom of the screen display. This is much better than other paging methods like DM1000 or Funnel Web Disk review to see single lines.

ONLY the SPACE BAR will pause the catalog until pressed again.

2015 added new keys to Directory: 1 = Editor.

A or a = Assembler file.

G or g = GPL Assembler file.

Use ENTER to select the filename so it will be placed into into a buffer, the cataloger will auto-load Dis/Fix 80 files into the EA3 menu, Programs will be EA5, and only Dis/Var 254 is considered to be XB programs. So to load XB programs use the SPACE BAR to buffer the filename, thus loading is automatic from there for XB programs. For DIS/VAR 80 or DIS/FIX 80 files to be edited or assembled use ENTER or SPACE BAR, then select the Edit or Assembler from the main menu. Loading is automatic from there.

Directory will automatically assume you wish to catalog a sub-directory if a Directory was selected. To buffer anything else you must use the SPACE BAR, to select a filename to be placed into a buffer, then auto return to REA main menu. Now select the option to be used from this buffer.

If you select D DIRECTORY again, the buffer will be used and the last device accessed will be used again. If you wish to clear the buffer just use FCTN BACK to the REA main menu.

## NOTE:

SOURCE file name must be filled in as this is the default. But if you use DIRECTORY to flag a file it will be placed into S SET PATH NAMES for all uses.

## . R X B

A previous feature that was optional since version 1000 but had no menu option on screen indicating it was a option. (Period) . will return to RXB menu screen.

## SYSTEM SUPPORT

The modified version of the Editor/Assembler no longer supports the 99/4 computer. A 99/4A is required. All TI BASIC support routines (CALL INIT, CALL LINK, CALL LOAD, CALL PEEK, CALL PEEKV, CALL POKEV, and CALL CHARPAT) have been removed from the Editor/Assembler. If you have a program that must be run from TI BASIC and requires these routines, you must plug an Editor/Assembler module into the cartridge connector. There are some assembly language programs that access data internal to the Editor/Assembler cartridge. These programs will not run correctly due to the re-structuring of the data in the Editor/Assembler module. For these programs you must use your Editor/Assembler cartridge. On the other hand like FunnelWeb REA loads the support routines before EA3 or EA5 loaders to engage, so C, FORTRAM, and FORTH will load from the EA5 prompt.

## NO 32K NEEDED TO WHAT?

REA has been totally re-written so the user can now use some of the features of REA without that nasty \*NO MEMORY EXPANSION\* error turning up. The error routine only disallows the user from accessing those aspects of REA that absolutely needs 32K to work. The user may now use the REA EDITOR PRINT FILE menu, or use the x R X B file loader menu, or use D DIRECTORY menu. That means with RXB and REA the user can now print files, view files, load any BASIC or XB program and catalog from REA with or without a 32K memory.

## EASTER EGGS

When on main menu of REA 2015 using keys 1 will still go to the Editor, 2 will still go to Assembler, 3 will still go to the Load and Run, and 5 will still go to RUN PROGRAM FILE. There are more to look for.