
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.
MOTION runs from ROM.

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 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 VERSION(numeric-variable)

Description

See EXTENDED BASIC MANUAL PAGE 190 for more data.
Also see Programs below.

| | |
|---|--|
| This line will ask for version and return current to numeric- variable X. | >CALL VERSION(X) |
| Line 100 asks for version num. | >100 CALL VERSION(V) |
| Line 110 checks for version to be larger than 2001 and if it is will ask for input to use a old routine CALL XB. | >110 IF V<2016 THEN INPUT "DSK NAME":D\$:: INPUT "FILENAME" :F\$:: CALL XB("DSK."&D\$&F\$) |
| | |

Options

Will always return current version of RXB. As you can see RXB
actually makes VERSION a valuable routine again.

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. Also RXB VGET is now in ROM.

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      RUN "XB"

             DELETE "XB"

             CALL CAT("XB")

             OLD XB

             SAVE XB      -(Must have a program within
                           -memory to work at all)

             CALL XB

```

Description

The XB DSR (Device Service Routine) allows access to the RXB title screen. The access will work only if the DSR is in the GPLDSR or LINK DSR. In other words, a DSR that acknowledges any type of DSR in RAM, ROM, GROM, GRAM, or VDP. Most DSR's only accept DSK or PIO. Others like the SAVE or LIST commands will only work with a program in the memory first. Still others like CALL LOAD("XB") must have the CALL INIT command used first.

From EA option 5 you may type XB then enter, or from EA option 3 type XB then enter, then enter again. If the EA option 1 (edit), then 4 (print) type XB. From TI BASIC use OLD XB or DELETE "XB".

Keep in mind that if it does not work, the problem is the DSR your using. Almost all DSR's today only acknowledge the ROM or RAM DSR's. As the XB DSR is in GROM/GRAM it seems a bit short sighted on the part of most programmers to use cut down versions of a DSR. Please discourage this as it is a disservice to us all.

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 too 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.

JOYSTICK and SPRITE POSITION CONTROL with KEY built in FIRE button

 As normal XB JOYSTICK and SPRITE controls were seperate 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 JOYMAP acts just like
 CALL JOYST + CALL KEY + CALL MOTION + CALL POSITION
 IF FIRE THEN line number

EXAMPLE:

CALL JOYMAP(key-unit,x-return,y-return,#sprite,row-index,
 col-index,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 MOTION but dot based
 dot-row,dot-column are like LOCATE
 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 BUFEEER 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.