artisan basic

# INTRODUCTION

ARTISAN BASIC is an extension of MSX BASIC. It is targeted at MSX1 machines with 64Kb memory and a disk system.

The idea for development came after competing in MSX BASIC competition.

<https://www.msxblog.es/concurso-msx-basic-9o-edicion/>

I have always felt that the capabilities of the machine could have been better exploited under BASIC.

The main areas that ARTISAN extension is focusing on are:

* Extended memory support
* Bitmap operations
* Animation support
* Sound player
* Helper functions

Extension loads itself in page 1 at address #4000 and provides several new commands. Following sections will describe main functionality groups and details about each new command are given later. Refer to the table of contents.

|  |  |  |
| --- | --- | --- |
| Version | Date | Description |
| 0.8 |  | Initial version |

# EXTENDED MEMORY SUPPORT

Standard MSX BASIC allows access to 32Kb of memory. In 64Kb systems there is another 32Kb hidden beneath ROM in pages 0 and 1. ARTISAN basic allows memory to be copied to and from this upper 32Kb. Additionally copying to and from VRAM can come from and to this upper 32Kb. Commands from other sections that take memory buffers as parameters can read data from this area of memory. There are also a few commands that allow copying data from and to VRAM.

Commands included are:

* [BOXMEMCPY](#_BOXMEMCPY)
* [BOXMEMVRM](#_BOXMEMVRM)
* [FILRAM](#_FILRAM)
* [FILVRM](#_FILVRM)
* MEMCPY
* MEMVRM
* VRMMEM

Since ARTISAN BASIC code also resides in this upper 32Kb, not all of it is free for use by programs. Memory map is given below:

Chart

Description automatically generated

ARTISAN BASIC does not occupy any memory below &H8000 allowing BASIC programs to have the same amount of free memory for code and variables as without the extension.

# BITMAP OPERATIONS

Several functions are provided to allow working with software sprites and tiling. Software sprites are defined by their data and mask that gets applied to background. Tiling functions allow placing data in a memory buffer or in video memory in a sequential fashion, when you want to apply one pattern over a larger area.

Commands included are:

* [BLIT](#_BLIT)
* TILEMEM
* TILEVRM

# ANIMATION SUPPORT

This section allows the creation of animation definitions that execute regularly based on VDP interrupt. Animation definitions allow changing of sprite pattern number, pattern data or changing character data.

To enable sprite animations, sprite handling has been revamped. Instead of PUT SPRITE commands one needs to define an array where sprite data is kept. This is transferred to VRAM on each interrupt.

Additionally grouping of sprites is supported which allow simultaneous moves and animation.

Commands in this section are grouped into several sections:

* Basic sprite handling system
  + SPRDISABLE
  + SPRENABLE
  + SPRSET
* Group of sprites handling
  + SPRGRPMOV
* Animation definitions
  + [ANIMITEMPAT](#_ANIMITEMPAT)
  + [ANIMITEMPTR](#_ANIMITEMPTR)
  + [ANIMDEF](#_ANIMDEF)
  + [ANIMSPRITE](#_ANIMSPRITE)
  + [ANIMCHAR](#_ANIMCHAR)
  + [AUTOSGAMDEF](#_AUTOSGAMDEF)
* Animation control
  + [ANIMSTART](#_ANIMSTART)
  + [ANIMSTOP](#_ANIMSTOP)
  + [ANIMSTEP](#_ANIMSTEP)
  + [AUTOSGAMSTART](#_AUTOSGAMSTART)
  + [AUTOSGAMSTOP](#_AUTOSGAMSTOP)
  + SGAM
* Animation memory buffers
  + MAXANIMDEFS
  + MAXANIMITEMS
  + MAXANIMSPRS
  + MAXAUTOSGAMS

## New sprite control system

The use of sprites is modified in ARTISAN basic in the following ways:

* Sprite attributes (location, pattern and color) are kept in an integer BASIC array of size (3,31)
* Values from the array are passed to VRAM during vertical blank if indicated by a specified integer variable
* Sprite control system is activated by SPRENABLE command
* When the system is active one should not run any commands that modify VRAM because of possible collision with sprite update
* When the system is active no new variables can be declared as this will cause corruption of the sprite control system

## Animation data memory handling

Defining animations requires some memory usage. This is located directly after the ARTISAN basic code in the segment &H4000-&h7FFF. That is why free memory in this segment depends on how many animations are defined. It is necessary to declare the maximum amount of each type of animation information before the use of definition commands. There are 4 types of definitions:

* Animation item – defines a single state
  + Sprite pattern, color and duration
  + Sprite/character pattern definition pointer and duration
* Animation definition – list of animation items to run
* Sprite/Character animation – link between which sprite/character to animate and with which animation definition
* Automatic Sprite Group Animation and Movement – automatic animation and movement between defined bounds of a sprite group

## BASIC program overall structure

The layout of the program that uses the sprite control system and animations is as follows:

* Declaration of all variables
* Declaration of sprite attributes array and the sprite update variable
  + SU%=0:DIM SA%(3,31)
* Reset of memory buffers for animations by defining zero size
* Resizing of memory buffers to required values
* Obtain free memory location in page 1 using MEMCPY(&H4010,VARPTR(A%),2) where A% was previously defined
* SPRENABLE (SA%,SU%,0/1)
* ON ERROR GOTO definition
* ON STOP GOSUB definition
* Main program
* On end/error/stop run:
  + Stop animations
  + SPRDISABLE

# SOUND PLAYER

ARTISAN basic includes the AKG player from ARKOS tracker

<https://www.julien-nevo.com/arkostracker/>

in version 2.01

Sound data should be exported from the Arkos tracker in binary format. Memory location can be in the first two memory pages.

Commands included in this section are:

* SNDPLYINI
* SNDPLYOFF
* SNDPLYON
* SNDSFX

# HELPER FUNCTIONS

This includes various functions that do not belong in previous sections and provide various functionality.

Commands included here are:

* [GENCAL](#_GENCAL)
* [COLL](#_COLL)

# ALPHABETICAL LIST OF COMMANDS

## ANIMCHAR

Defines single character animation sequence.

Format:

ANIMCHAR (byte ID, integer character\_number, byte animation\_definition\_id, byte cyclic\_flag)

Where ID is between 0 and MAXANIMSPRS value

Character\_number specifies the character to animate (0-767)

Animation\_definition\_id is between 0 and MAXANIMDEFS

Cyclic\_flag of 0 means that the animation will run one time only, other values mean a looping animation

Prerequisites:

* MAXANIMSPRS reserved memory for definition
* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMCHAR(0,255,0,1)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMDEF

Defines a list of animation items which is later associated with a character or a sprite.

Format:

ANIMDEF (byte ID, byte size, integer[] values)

Where ID is between 0 and MAXANIMDEFS value

Size is number of animation items (1-15)

Values holds animation item IDs that form this animation definition

Prerequisites:

* MAXANIMDEFS reserved memory for definition
* Animation items prepared with ANIMITEMPTR/ANIMITEMPAT

Errors:

* Invalid type if incorrect type passed
* Subscript out of range if ID invalid
* Overflow if size outside 1-15 range
* Index out of bounds if values array smaller than size parameter

Example:

DIM V%(1):V%(0)=0:V%(1)=1

\_ANIMDEF(0,2,V%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMITEMPAT

Defines a single animation state where sprite pattern and color are specified. Usable for sprites only.

Format:

ANIMITEMPAT (byte ID, integer ticks, byte pattern, byte color)

Where ID is between 0 and MAXANIMITEMS value

Ticks is number of interrupts that this animation item lasts before stepping over to the next state as defined in animation definition (>0)

Pattern specifies sprite pattern to apply to a sprite

Color specifies the color to apply to a sprite

Prerequisites:

* MAXANIMITEMS reserved memory for definition

Errors:

* Subscript out of range if ID invalid
* Overflow if ticks=0

Example:

\_ANIMITEMPAT(0,4,5,6)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMITEMPTR

Defines a single animation state where pattern data is specified. Applicable to sprites and characters.

Format:

ANIMITEMPTR (byte ID, integer ticks, integer pointer)

Where ID is between 0 and MAXANIMITEMS value

Ticks is number of interrupts that this animation item lasts before stepping over to the next state as defined in animation definition (>0)

Pointer is a memory location where pattern data is located, can be in pages 0 and 1.

Prerequisites:

* MAXANIMITEMS reserved memory for definition

Errors:

* Subscript out of range if ID invalid
* Overflow if ticks=0

Example:

\_ANIMITEMPTR(1,3,&H2000)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMSPRITE

Defines single sprite animation sequence.

Format:

ANIMSPRITE (byte ID, integer sprite\_number, byte animation\_definition\_id, byte cyclic\_flag)

Where ID is between 0 and MAXANIMSPRS value

sprite\_number specifies the sprite to animate (0-31)

Animation\_definition\_id is between 0 and MAXANIMDEFS

Cyclic\_flag of 0 means that the animation will run one time only, other values mean a looping animation

Prerequisites:

* MAXANIMSPRS reserved memory for definition
* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMSPRITE(0,5,0,1)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMSTART

Starts animation sequence.

Format:

ANIMSTART (byte ID)

or

ANIMSTART (byte item\_number, integer[] sprite\_animations)

Where ID is between 0 and MAXANIMSPRS value

item\_number specifies the number of animations in the array

sprite\_animations array holds animation ids to start simultaneously

Prerequisites:

* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMSTART(1)

Or

DIM A%(2):A%(0)=0:A%(1)=1:A%(2)=2

\_ANIMSTART(3,A%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMSTEP

Manually progresses animation which is not started with ANIMSTART.

Format:

ANIMSTEP (byte ID)

or

ANIMSTEP (byte item\_number, integer[] sprite\_animations)

Where ID is between 0 and MAXANIMSPRS value

item\_number specifies the number of animations in the array

sprite\_animations array holds animation ids to step simultaneously

Prerequisites:

* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMSTEP(1)

Or

DIM A%(2):A%(0)=0:A%(1)=1:A%(2)=2

\_ANIMSTEP(3,A%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMSTOP

Stops animation sequence.

Format:

ANIMSTOP (byte ID)

or

ANIMSTOP (byte item\_number, integer[] sprite\_animations)

Where ID is between 0 and MAXANIMSPRS value

item\_number specifies the number of animations in the array

sprite\_animations array holds animation ids to stop simultaneously

Prerequisites:

* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMSTOP(1)

Or

DIM A%(2):A%(0)=0:A%(1)=1:A%(2)=2

\_ANIMSTOP(3,A%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## AUTOSGAMDEF

Defines automatic sprite group animation and movement between specified bounds.

Format:

AUTOSGAMDEF (byte ID, integer variable x, integer variable y, integer minimum, integer maximum, integer delta, integer direction, integer ticks, byte sprite\_group\_size, integer[2][] variable sprite\_group, byte item\_number, integer[] variable sprite\_animations\_negative\_direction, integer[] variable sprite\_animations\_positive\_direction )

Where ID is between 0 and MAXAUTOSGAMS value

X is integer variable that holds horizontal sprite group location

Y is integer variable that holds vertical sprite group location

Minimum is the low range value of possible locations

Maximum is the high range value of possible locations

Delta is the step value for movement

Directions defines horizontal (=0) or vertical (!=0) direction

Ticks is the number of interrupts between sprite group movement and stepping through animations

Sprite\_group\_size defines number of sprites in a sprite group

Sprite\_group is an array describing a sprite group, for details refer to SGAM command

Item\_number defines number of animations to step through

Sprite\_animations\_negative\_directions holds animations for when sprite group is going backwards

Sprite\_animations\_positive\_directions holds animations for when sprite group is going forward

Prerequisites:

* MAXAUTOSGAMS reserved memory for definition
* Animations prepared with ANIMSPRITE

Errors:

* Invalid type if incorrect type passed
* Subscript out of range if ID invalid

Example:

DIM AL%(2):AL%(0)=0:AL%(1)=1:AL%(2)=2

DIM AR%(2):AR%(0)=3:AR%(1)=4:AR%(3)=5

DIM SG%(2,1):SG%(0,0)=0:SG%(1,0)=0:SG%(2,0)=0

SG%(0,1)=1:SG%(1,1)=0:SG%(2,1)=0

X%=0:Y%=0

\_AUTOSGAMDEF(0,X%,Y%,0,100,1,0,1,2,SG%,3,AL%,AR%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## AUTOSGAMSTART

Starts automatic sprite group movement and animation.

Format:

AUTOSGAMSTART (byte ID)

Where ID is between 0 and MAXAUTOSGAMS value

Prerequisites:

* Animation definition prepared with AUTOSGAMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_AUTOSGAMSTART(1)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## AUTOSGAMSTOP

Stops automatic sprite group movement and animation.

Format:

AUTOSGAMSTOP (byte ID)

Where ID is between 0 and MAXAUTOSGAMS value

Prerequisites:

* Animation definition prepared with AUTOSGAMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_AUTOSGAMSTOP(1)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## BLIT

Command implements software sprite functionality. It apply monochrome object of defined size onto defined memory background with 1 pixel precision. Object is defined with mask and data. Mask will be ANDed with background and then data will be ORed with background. All memory locations can be in pages 0 and 1.

Format:

BLIT (integer x, integer y, integer object\_data\_pointer, integer object\_mask\_pointer, integer width, integer height, integer background\_pointer, integer background\_width, integer background\_height)

Where

X is location in the background (>=0)

Y is location in the background (>=0)

Object\_data\_pointer is a memory location where object foreground is defined

Object\_mask\_pointer is a memory location where object mask is defined

Width is object width in characters (8 pixels)

Height is object height in characters (8 pixels)

Background\_pointer is a memory location where background is located

Background\_width is background width in characters (8 pixels)

Background\_height is background height in characters (8 pixels)

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_BLIT(55,31,&h7000,&h7800,12,5,&h100,32,24)

Sample code:

* DEMO2.BAS

## BOXMEMCPY

Copies window like data segment from one location into another. Locations can be in pages 0 and 1.

Diagram

Description automatically generated

Format:

BOXMEMCPY (integer P1, integer B3, integer number\_of\_rows, integer B1, integer P2, integer B2)

Where

P1 is memory location where source data begins

B3 is number of bytes in a single row of source data

Number\_of\_row is number of rows of source data

B1 is number of bytes of a source window row

P2 is memory location where to copy data

B2 is number of bytes of destination window row

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_BOXMEMCPY(&H1000,80,256,5,&H7000,80)

Sample code:

* DEMO2.BAS

## BOXMEMVRM

Copies window like data segment from one location in RAM into another in VRAM. Source location can be in pages 0 and 1. Command parameters are the same as for BOXMEMCPY. B2 value should be 256 for SCREEN 2 mode.

Format:

BOXMEMVRM (integer P1, integer B3, integer number\_of\_rows, integer B1, integer P2, integer B2)

Where

P1 is memory location where source data begins

B3 is number of bytes in a single row of source data

Number\_of\_row is number of rows of source data

B1 is number of bytes of a source window row

P2 is memory location where to copy data

B2 is number of bytes of destination window row

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_BOXMEMVRM(&H1000,80,256,5,BASE(12),256)

Sample code:

* DEMO2.BAS

## COLL

Collision detection between one rectangular object and a list of other rectangular objects.

Format:

COLL (integer variable result, integer x, integer y, integer width, integer height, integer list\_size, integer[7][] objects)

Where

Result is an integer variable where the result is stored, -1 if no collision, 0..list\_size-1 if collision

X is horizontal location of upper left edge

Y is vertical location of upper left edge

Width is the last column of an object, for a 16x16 sprite this is 15

Height is the last row of an object, for a 16x16 sprite this is 15

List\_size is the number of objects to check collision agains and stored in objects variable

Objects is a two dimensional array that describes collidable objects. These can either be static or sprites. For of a single array element is:

(0,n) – active flag, if 0 collision will not be checked

(1,n) – is horizontal location of upper left edge OR sprite ID depending on (7,n)

(2,n) – is horizontal location of upper left edge OR not used depending on (7,n)

(3,n) – horizontal offset where actual object begins, for example if a sprite pattern does not actually begin at (0,0)

(4,n) – vertical offset where actual object begins, for example if a sprite pattern does not actually begin at (0,0)

(5,n) – width or the last column of the object

(6,n) – height or the last row of the object

(7,n) – type, 0=generic, <>0 sprite

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

(X% and Y% already defined)

R%=0:DIM O%(7,1)

O%(0,0)=1:O%(1,0)=100:O%(2,0)=80:O%(3,0)=0:O%(4,0)=0:O%(5,0)=9:O%(6,0)=9:O%(7,0)=0

O%(0,1)=1:O%(1,1)=31:O%(3,1)=4:O%(4,1)=4:O%(5,1)=5:O%(6,1)=5:O%(7,1)=1

\_COLL(R%,X%,Y%,15,15,2,o%)

Sample code:

* COLLISION.BAS
* GAME.BAS

## FILRAM

Fills memory block with a specified value. Can be used for pages 0 and 1.

Format:

FILRAM (integer address, integer count, byte value)

Where

Address is the starting memory block location

Count is the number of bytes to write

Value is the number to fill the block with

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_FILRAM (&h1000,1024,0)

Sample code:

* None

## FILVRM

Fills video memory block with a specified value.

Format:

FILVRM (integer address, integer count, byte value)

Where

Address is the starting video memory block location

Count is the number of bytes to write

Value is the number to fill the block with

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_FILVRM (BASE(12),6144,0)

Sample code:

* BLIT.BAS

## GENCAL

Generic assembly call. Allows specifying registers AF, BC, DE, HL, IX and IY before calling specified address. Resulting register values are store back in the input array. Routine does not put RAM in pages 0 and 1 so one can call BIOS routines.

Format:

GENCAL (integer address, integer[5] registers)

Where

Address is the location of the routine to call

Registers in an array holding input and output register values. Order of registers in the array is: AF, BC, DE, HL, IX , IY

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if register array too short

Example:

REM COPY MSX FONT TO VRAM IN SCREEN 2

DIM R%(5)

R%(1)=2048:R%(2)=256\*PEEK(5)+PEEK(4):R%(3)=BASE(12)

\_GENCAL(&H5C,R%)

Sample code:

* NONE