Atari 8 Bit Action! Library Reference

Atari 8 Bit Action! Library Reference

Library Version 1.51 (2022.08.27)

Reference Revision D

Wade Ripkowski
inverseatascii@icloud.com

Table of Contents

Gadgets (LIBGADG.ACT) 15

Menus (LIBMENU.ACT) 24

Input/Output (LIBIO.ACT) 26

Serial Input/Output (LIBSIO.ACT) 28

String Manipulation (LIBSTR.ACT) 29

Overview

The library described herein is designed for use with the Action! programming language by Optimized Systems Software (OSS) for the Atari 8 bit home computer.

The library was initially written in 2015 and included only the base windowing system. Over time it was expanded to include general purpose routines, and gadgets (which are windowing system add-ons).

The major features offered by the library are:

- Window System
  - The window management system allows the programmer to open and close windows with different styles. To reduce complexity and overhead it is a LIFO (last in first out) design. It is intended for the programmer to keep track of the call stack.
- Gadgets
  - Gadgets are windowing system add-ons which are designed to provide simple things like alert boxes, progress bars, and input controls.
- Menus
  - Menus are a windowing system add-on which are designed to provide menu controls.
- Input/Output
  - The input/output routines pick up where the Action! and Action! Toolkit routines leave off, such as reading two bytes at a time. Also included are variables and a routine for setting up the SIO DCB and calling the SIO vector.
- String Manipulation
  - Functions to aid with string manipulation and character conversion.
- DOS Functions
  - Functions for interacting with DOS.
- Miscellaneous
  - Helper functions that don't fall into any particular category, including waiting with and without keystrokes.

References in this documentation that refer to `void` are meant to mean not applicable, and not a data type. Other data types are described as they are defined by Action!.

Symbol Table Warning

When using individual sections of the library, you typically will not need to do anything special other than include the file(s) at the top of your applications source, and compile.

When using multiple sections of the library at once, you will need to ensure there is an adequate symbol table size and symbol space available. When all of the library sections are included in one application, while the symbol table may not be full, it will run out of space.

By default Action! reservers room for 255 symbol table entries. If you get an error 3, 4, or 61 when compiling your program, it most likely means the symbol table space is insufficient. This can be overcome rather easily. OSS made provisions in Action! to accommodate a larger symbol table.

To increase the symbol table size in Action!, you will need cold boot, then load `BIGST.ACT`. This will expand the symbol table to a maximum of 510 entries. Before compiling, edit the file. Look for the line:

```
  bigST = 'm
```

Change the value from `m` to `D`. This value is the break point at which the expanded symbol table is broken into two segments. The value used is subjective and may need to be different based on your application and the variable names used - read the documentation and also the notes in `BIGST.ACT` itself. `D` works well for my coding style which primarily uses Hungarian notation for variable names, and Uppercase function names.

Once changed, compile and run. Immediately following the run, you can try to compile your program. If you get an error 61, you also need to increase the symbol table space. By default Action! reserves 2K (eight 256 byte pages) of memory for symbol table space.

To add symbol table space, immediately after running `BIGST.ACT`, enter the Action! monitor, and execute the following, which will increase the symbol table space from the default 8 pages to 12 pages. Depending on your program you may need even more:

```
SET $495=12
```

The value 12 works when compiling the stub programs included with this library. Not all stub programs need the big symbol table. Those that do will have a comment at the top stating the need.

The big symbol table changes remain in effect until the Action! cartridge is rebooted.

For more in depth explanation on how Action! uses the symbol table, see the Action! runtime reference section VII.

Code Size

Management

When using all of the components of the library, the code size of your application could start to become rather large. If you find your program no longer fits in available memory or does not have enough memory for variables after loading, you may need to optimize the compile environment.

To optimize the code size, copy all of the library files to your project directory. Subsequently modify your applications source files to include the library files from this location rather than the original library source location.

Now that your application is including the library from your application project location, you can proceed. You will want to cross reference functions defined by the library with those your application uses, including dependencies of the library functions (some library functions call others).

One you have identified all of the library functions (and their dependents) used by your application, you will then remove modify the library files (in your project directory, NOT the original source). In these library files, you will remove any functions that are not needed by your application, thus reducing the overall compiled code size.

This can be extended to include the Action! runtime package if you so desire. Be careful removing functions from the Action! runtime library file, because the compiler will backfill unresolved references to the ROM cartridge counterparts, which will prevent the executable from running without the cartridge.

Compilation

When compiling a large application, you may run into problems that are related to source code size, not object code size. If the editor has a large amount of source in it, Action! may not have enough room to execute the compile. In this case you will want to compile it from disk with the editor contents empty.

If you want to create a stand alone executable, you will need to include the Action! runtime package in your build. This is done by either including “SYS.ACT” or the individual Action! runtime libraries. SYS.ACT includes all of the runtime library.

You may also need to set your applications memory load address and execution address (init vector). Safe locations will vary by application and DOS, and may take some experimentation.

Requirements

Action! Runtime Library Dependency

This library depends upon some of the Action! runtime library functions. Library routines will list the runtime function dependencies in the API reference which follows in this documentation.

Routines from the Action! Runtime Library that are needed:
GetD()
MoveBlock()
Poke()
Position()
Put()
PutD()
SCopy()
SCopyS()
SetBlock()
Zero()

Symbol Table

The symbols used by this library are as follows. Many of the names are re-used throughout the library, and are kept to a short length to converse space.

bC
bCAP
bD
bE
bHLP
bI
bINV
bK
bL
bN
bP
bR
bRCH
bRCO
bS

bT
cB
cD
cE
cL
cS
iV
pA
pS
x
xp
y
yp

baW

cpWM
pWn
vCur

RTCLK
DAUX1
DAUX2
DBUF
DBYT
DCOMND
DDEVIC
DSTATS
DTIMLO
DUNIT

File Reference

DEFINES.ACT

All definitions used throughout the library.

This should be included FIRST at the top of the main program file, and should be included in any program that uses the library routines.

DEFWIN.ACT

Window type definitions and variables used by the window system portion of the library.

If the windowing system is used, this file should be included immediately after DEFINES.ACT, and BEFORE LIBWIN.ACT.

LIBDOS.ACT

Collection of DOS related functions.

LIBGADG.ACT

Collection of gadgets (add-ons) for the window system.

When using these routines, LIBWIN.ACT MUST be included before.

LIBIO.ACT

Collection of Input and Output routines that augment the Action! and Action! Toolkit routines.

LIBMENU.ACT

Collection of menu routines which simplifies program navigation.

When using these routines, LIBWIN.ACT MUST be included before.

LIBMISC.ACT

Collection of routines that don’t fall into the other categories.

LIBSIO.ACT

OS SIO DCB variables and SIO vector wrapped as a procedure, to enable direct serial Input and Output per the SIO bus.

LIBSTR.ACT

Collection of string manipulation routines that augment the Action! and Action! Toolkit routines.

LIBWIN.ACT

Collection of window routines that make up the text window system.

When using these routines, `DEFINES.ACT`, `DEFWIN.ACT`, and `LIBSTR.ACT` MUST be included before.

API Reference

Window System (LIBWIN.ACT)

void PROC WBack(byte bN)

Parameters: bN = Internal code of character
Returns: void
Requires: DEFWIN.ACT
Runtime - SetBlock()

Description
Sets the background “image” that covers the entire screen. This is a single character to repeat in every cell.

Using large footprint characters (a lot of pixels) can make the program elements like windows and menus harder to see. It is best used with small footprint characters like the ‘.’. With a custom character set, this function could be advantageously used.

byte FUNC WClose(byte bN)

Parameters: bN = Window handle number
Returns: byte = 0 for success
-or-
>100 on error (default)
Requires: DEFINES.ACT
DEFWIN.ACT
Runtime - MoveBlock(), Zero()

Description
Closes an open window specified by the handle bN.

If window is not open, no action is taken.

It is up to the programmer to close windows in the proper order - the last one opened should be the first one closed. If an earlier window is closed before a more recent overlapping window, the screen contents will not be reflected accurately when the latter is closed (it will show remnants of the earlier window).

byte FUNC WClr(byte bN)

Parameters: bN = Window handle number
Returns: byte = Success status
0 = Succesful
WENOPN = Window not open (default)
Requires: DEFINES.ACT
DEFWIN.ACT
LIBSTR.ACT - StrInv()
Runtime - MoveBlock(), SetBlock()

Description
Clears the contents of the window referenced by window handle bN. Effectively clearing the screen of the windows interior dimensions (excluding frame).

byte FUNC WDiv(byte bN, y, bD)

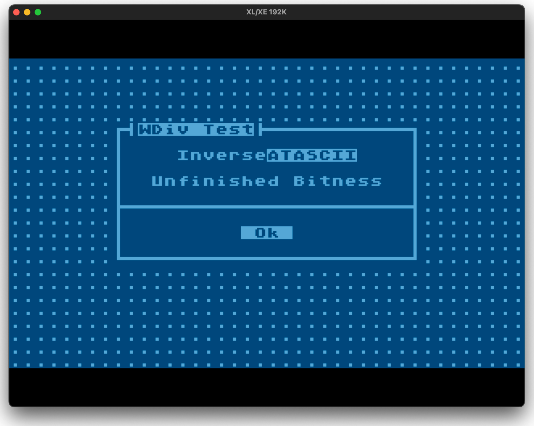
Parameters: bN = Window handle number
y = Window row to display divider
bD = On/Off flag
WON = On (show divider)
WOFF = Off (remove divider)
Returns: byte = Success status
0 = Succesful
WENOPN = Window not open (default)
Requires: DEFINES.ACT
DEFWIN.ACT
Runtime - MoveBlock(), SCopy()

Description
Draws a divider line in the window referenced by handle bN.

The divider is drawn on row y of the window.

The bD (display on/off) parameter is passed as WON, the bar will be displayed. With WOFF, the bar will be removed which will blank the contents on the window row and restore the window frame.

Calling WDiv() with WOFF is also a quick way to clear one line of a window.



InverseATASCII
Unfinished Bitness

void PROC WInit()

Parameters:	`void`	
Returns:	`void`	
Requires:	`DEFINES.ACT`	
	`DEFWIN.ACT`	
	`Runtime`	`- Poke(), Position(), Put(), Zero()`

Description
Used to initialize the window management system. It should be called before any other windowing system call.

In addition to defaulting all the windowing system variables, it will perform the following:

- Turn the cursor off (poke 752, 1)
- Set the left screen margin to 0 (poke 82, 0)
- Set the cursor position to the top left corner (0,0)
- Clear the screen

The library is built to handle 10 windows. You can alter this routine for more or less as your program requires. Memory requirements will increase or decrease as the number is changed. Increasing the number may also necessitate increasing the window system storage space by increasing the value of `WBUFSZ` in file `DEFWIN.ACT`.

byte FUNC WOpen(byte x, y, w, h, bT)

Parameters: x = Column of screen for left edge of window
y = Row of screen for top edge of window
w = Width of window in columns
h = Height of window in rows
bT = Inverse video flag (optional)
WON = Inverse video
WOFF = Normal video (default)

Returns: byte = Window handle number
-or-
>100 on error

Requires: DEFINES.ACT
DEFWIN.ACT
Runtime - MoveBlock(), SetBlock()

Description
Opens a window on the screen with a single line border. The screen contents under the window are saved, then restored when the window is closed.

Top left coordinate is specified by x and y. The width and height are specified with w and h. If the inverse flag, bT, is set, the window is drawn and filled in inverse video.

byte FUNC WOrn(byte bN, bT, bL char pointer pS)

Parameters: bN = Window handle number
bT = Top or bottom of window designation
WPTOP = Top border
WPBOT = Bottom border
bL = Left, right, or center of window designation
WPCNT = Center
WPLFT = Left side
WPRGT = Right side
pS = Pointer to character string of title text
Maximum size is 36 characters!
Returns: byte = Success status
0 = Succesful
WENOPN = Window not open (default)
Requires: DEFINES.ACT
DEFWIN.ACT
LIBSTR.ACT - StrAI(), StrInv()
Runtime - MoveBlock(), SCopy()

Description

Sets a window ornament to text string s with decorations on the window referenced by bN, on either the top or bottom border as given by bT, and left or right side as given by bL.

If an ornament is to be set, the window itself must be large enough to accommodate it, along with any other assigned ornaments. For a single ornament, a minimum window width should be the title length plus four characters (two characters for the ornaments on either side of the tile, and two characters for the window frame where the ornaments can't be drawn). Because of this, the maximum length of a title is 36 characters.

If multiple ornaments are used on top or bottom at the same time, care must be taken to ensure the window size is large enough, or the ornament size is small enough, to accommodate both ornaments.

byte FUNC WPos(byte bN, x, y)

Parameters: bN = Window handle number
x = Window column to move cursor to
y = Window row to move cursor to
Returns: byte = Success status
0 = Succesful
WENOPN = Window not open
Requires: DEFINES.ACT
DEFWIN.ACT
Runtime - Position()

Description
Moves the window systems virtual cursor to the screen position of the specified x and y coordinates within the window referenced by window handle bN.

byte FUNC WPrint(byte bN, x, y, char pointer pS)

Parameters: bN = Window handle number
x = Window column to print text
y = Window row to print text
pS = Pointer to character string of text to print
Maximum size is 38 characters!
Returns: byte = Success status
0 = Succesful
WENOPN = Window not open (default)
Requires: DEFINES.ACT
DEFWIN.ACT
LIBSTR.ACT - StrAI(), StrInv()
Runtime - MoveBlock(), SCopy()

Description
Prints text string pointed to by pS at the virtual cursor position of x and y within the window referenced by window handle bN.

A minimum window width should be the text length plus two characters (for the window frame). Because of this, the maximum length of a text is 38 characters.

byte FUNC WPut(byte bN, x)

Parameters: bN = Window handle number
x = Character to put
Returns: byte = Success status
0 = Succesful
WENOPN = Window not open
Requires: DEFINES.ACT
DEFWIN.ACT
LIBSTR.ACT - StrAI()
Runtime - MoveBlock()

Description
Outputs the character specified by x at the window systems virtual cursor within the window referenced by window handle bN.

Increments the window systems virtual cursor by one column.

If the window was created with the inverse flag set, the character will be inversed to match.

byte FUNC WStat(byte bN)

Parameters: bN = Window handle number
Returns: byte = Window status
WON = In use (window ON)
WOFF = Not in use (window OFF)
Requires: DEFWIN.ACT

Description
Returns the status of the window specified by the handle bN.

`byte FUNC WTitle(byte bN card pointer pS)`

*** **DEPRECATED** ***

Parameters: bN = Window handle number
pS = Pointer to character string of title text
Maximum size is 36 characters!
Returns: byte = Success status
0 = Succesful
WENOPN = Window not open (default)
Requires: DEFINES.ACT
DEFWIN.ACT
LIBSTR.ACT - StrAI(), StrInv()
Runtime - MoveBlock(), SCopy()

Description
Sets the window title to s with ornaments for the window referenced by bN.

This is a deprecated function, replaced by WOrn().
Calling WTitle() is the same as calling WOrn() with WPTOP and WPLFT set for positioning.

Gadgets (LIBGADG.ACT)

void PROC GAlert(char pointer pS)

Parameters: pS = Pointer to character string to display
Maximum size is 38 characters!

Returns: void

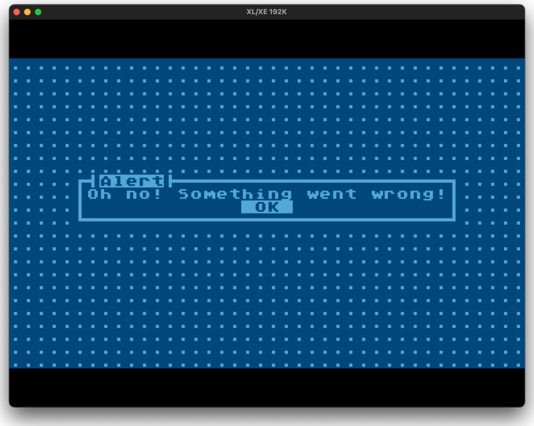
Requires: DEFINES.ACT
DEFWIN.ACT
LIBWIN.ACT - WOpen(), WTitle(), WPrint(), WClose()
LIBMISC.ACT - WaitKC()

Description

Displays a screen centered modal window with the title "Alert" and the message text of the string pointed to by char pointer pS. It will display an OK "button" beneath the text and wait for keystroke, which will be consumed.

Calling GAlert will consume one window handle while it is open.

Because the window will have a frame, the maximum message length is 38 characters.

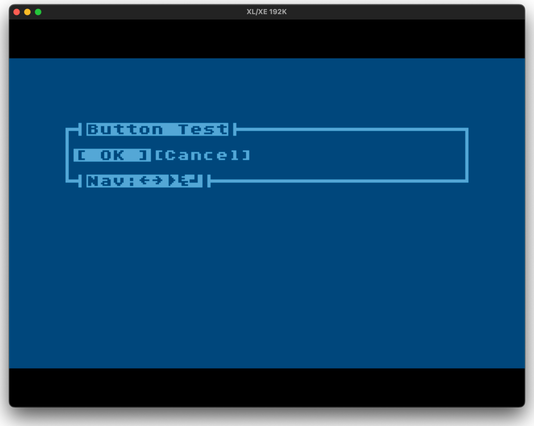


Oh no! Something went wrong!

byte FUNC GButton(byte bN, x, y, bD, bS card pointer pA)

Parameters: bN = Window handle number
x = Window column to start get
y = Window row
bD = Initial selected button
bS = Number of buttons in array
pA = Pointer to ragged array of button name strings
Returns: byte = Button number selected or XESC (escape exit) or XTAB (tab exit)
Requires: DEFINES.ACT
DEFWIN.ACT
LIBWIN.ACT - WPrint()
LIBSTR.ACT - StrInv()
LIBMISC.ACT - WaitKC()
Runtime - SCopy()

Description
Displays a row of buttons and gets selection from user.



If the initial selection indicator (bD) is passed as GDISP, then the buttons will be displayed and the function will exit (none will be highlighted).

It is up to the programmer to define the button ornaments, if any. For example: [OK]. In this example the [and] are the ornaments enclosing the 4 character string space OK space. The entire string will be inversed when selected, including the ornaments.

Care must be taken on the total length of the button strings contained in ragged array pointer pA. The total should be no more than 38 for a window that is 40 wide.

Keys accepted are:

LEFT\+	= Move button selector left
RIGHT*	= Move button selector right
UP\-	= Move button selector left
DOWN\=	= Move button selector right
ESCAPE	= Exits without selection (returns XESC)
TAB	= Exits without selection (returns XTAB)
ENTER	= Accepts current selected button and exits (returns selected button #)

byte FUNC GCheck(byte bN, x, y, bI, bD)

Parameters: bN = Window handle number
x = Window column to start get
y = Window row
bI = Display Only indicator
GDISP (0) to display and exit
bD = Default initial value
GCON = Checked
GCOFF = Unchecked

Returns: byte = Checked status as GCON or GCOFF or XESC (escape exit) or XTAB (tab exit)

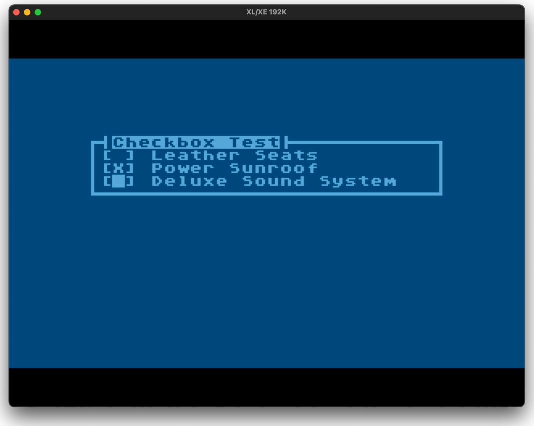
Requires:
```
DEFINES.ACT
DEFWIN.ACT
LIBWIN.ACT   - WPrint()
LIBMISC.ACT  - WaitKC()
```

Description

Displays a checkbox ([ ]) and gets selection from user.

Unlike many other input gadgets, the text for the option is not included and should be displayed separately in the window using WPrint() prior to calling GCheck().

When the checkbox is marked, an inverse video X will be displayed, otherwise it will be an inverse space. When the function exits, the set value will be displayed in normal video. ENTER must be used to set (lock) the value to be returned, otherwise the default value passed in is re-displayed.



```
Checkbox Test
[ ] Leather Seats
[X] Power Sunroof
[█] Deluxe Sound System
```

If the display only indicator (bI) is passed as GDISP, then the checkbox will be displayed and the function will exit. Display Only will respect default values and represent them accordingly. This is useful for drawing the checkbox on a form before selection is to occur.

Keys accepted are:

ESCAPE	= Exits without selection (returns XESC)
TAB	= Exits without selection (returns XTAB)
SPACE	= Toggle value of checkbox (display only)
X/x	= Acts just like SPACE
ENTER	= Accepts (sets and locks to displayed current value) and exits

byte FUNC GInput(byte bN, x, y, bT, bS char pointer pS)

Parameters:
- bN = Window handle number
- x = Window column to start get
- y = Window row
- bT = Allowed character type
  - GANY = Any non-cursor control character
  - GALNUM = Any Alpha-Numeric character (0-9, a-z, A-Z, <space>)
  - GALPHA = Alphabetic characters only (a-z, A-Z, <space>)
  - GNUMER = Numeric characters only (0-9, ., -)
- bS = Display size for string (max 38)
- pS = Pointer to text string to input/edit

Returns:
- byte = Success indicator
  - TRUE = String was modified
  - FALSE = String was not modified

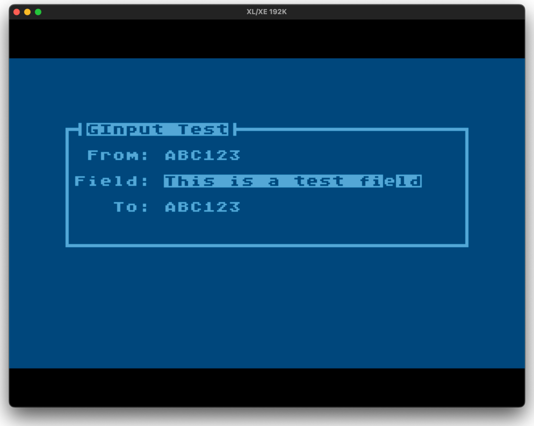
Requires:

```
DEFINES.ACT
DEFWIN.ACT
LIBSTR.ACT  - StrInv()
LIBMISC.ACT - WaitKC(), IKC2ATA()
LIBWIN.ACT  - WPrint()
Runtime     - SCopy(), SCopyS(), SetBlock()
```

Description
Edits a large string in a smaller display window by scrolling through the string and displaying only a portion at a time, much like modern operating system input fields.

The edit area is opened in the window handle referenced by bN. The edit area is placed at the x and y position in the window. The maximum size of the edit area is specified by bS, and the maximum should be considered to be 38 (given a window that is 40 characters wide).

The initial edit area contents will be a copy of the string passed as pS.



```
 GInput Test
  From: ABC123
 Field: This is a test field
    To: ABC123
```

If the input is exited using ESC, the string passed will be left in tact. If the input is exited using the ENTER key, any edits made will be copied to the string passed via pointer pS. This means you can not pass a static text string such as “Hello World”, it MUST be CHAR ARRAY or CHAR POINTER.

Keys accepted are:

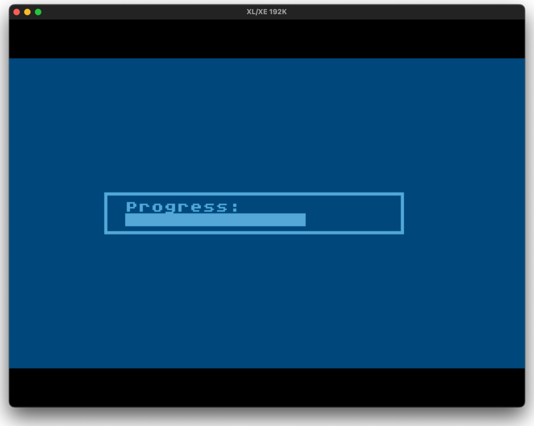
`LEFT\+`	= Move cursor left
`RIGHT\*`	= Move cursor right
`DEL`	= Delete character left of cursor (or 1st char if cursor is at position 1)
`Control-DEL`	= Delete character at cursor (move remainder left 1 position, add space at end)
`Shift-DEL`	= Delete entire string contents (moves cursor to position 1 of text string)
`INSERT`	= Insert space at cursor (character at end of text string will be lost)
`Control-Shift-S`	= Move cursor to beginning of string
`Control-Shift-E`	= Move cursor to end of string
`ESCAPE`	= Cancel edits and exit
`ENTER`	= Accept edits and exit

`void PROC GProg(byte bN, x, y, bS)`

Parameters: bN = Window handle number
x = Window column to display bar at
y = Window row to display bar at
bS = Bar size (Percent complete)
Returns: `void`
Requires:

```
DEFINES.ACT
DEFWIN.ACT
LIBWIN.ACT   - WPrint()
Runtime      - SCopy()
```

Description
Displays a progress bar at the x and y position within the window referenced by window handle bN. The percentage complete is referenced by bS.



byte FUNC GRadio(byte bN, x, y, bD, bI, bS card pointer pA)

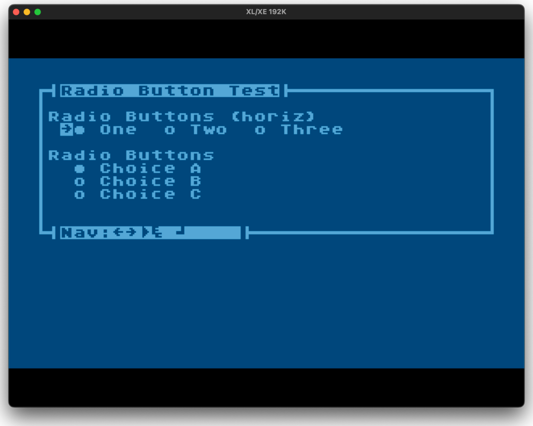
Parameters:
- bN = Window handle number
- x = Window column to start get
- y = Window row
- bD = Direction of button placement
  - GHORZ = Horizontal (side by side)
  - GVERT = Vertical (stacked)
- bI = Initial selected button
  - GDISP (0) to display and exit
- bS = Number of buttons in array
- pA = Pointer to ragged array of button name strings

Returns:
- byte = Button number selected or XESC (escape exit) or XTAB (tab exit)

Requires:
```
DEFINES.ACT
DEFWIN.ACT
LIBWIN.ACT   - WPrint(), WPos(), WPut()
LIBSTR.ACT   - StrInv()
LIBMISC.ACT  - WaitKC()
```

Description

Displays a selection of radio buttons and gets a selection of one from user.



```
┤Radio Button Test├
Radio Buttons (horiz)
 →● One  o Two  o Three

Radio Buttons
  ● Choice A
  o Choice B
  o Choice C

┤Nav:←→ ⏎├
```

Only one button from the defined group can be selected. When there is a need for multiple option selection GCheck() should be used instead.

The buttons will be arranged in the direction specified by bD. Valid directions are GHORZ or horizontal (side by side), or GVERT for vertical (stacked) alignment. Care should be taken to ensure the window boundaries are large enough to accommodate the buttons, especially when aligning horizontally. For horizontal buttons, it is only reasonably to expect 3 or 4 buttons to fit in the 38 columns available inside a window frame. Each horizontal button is separated by 2 spaces. For this reason, it is recommended to use vertical alignment (GVERT) to stack the buttons for more than 3 buttons.

If the initial selection indicator (bI) is passed as GDISP, then the buttons will be displayed and the function will exit (none will be highlighted). This is useful for drawing the buttons on a form before selection is to occur.

Keys accepted are:

`LEFT\+`	= Move button selector left
`RIGHT\*`	= Move button selector right
`UP\-`	= Move button selector left
`DOWN\=`	= Move button selector right
`ESCAPE`	= Exits without selection (returns `XESC`)
`TAB`	= Exits without selection (returns `XTAB`)
`SPACE`	= Set currently selected button as choice
`ENTER`	= Accepts current selected button and exits (returns selected button #)

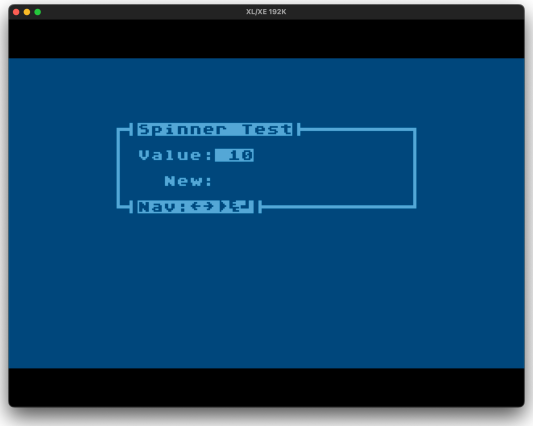
byte FUNC GSpin(byte bN, x, y, bL, bM, bP)

Parameters:	bN = Window handle number
	x = Window column to display value at
	y = Window row to display value at
	bL = Lowest allowed value
	bM = Maximum allowed value
	bP = Present (current) value
Returns:	byte = value selected
Requires:	DEFINES.ACT
	DEFWIN.ACT
	LIBWIN.ACT - WPrint()
	LIBSTR.ACT - StrPad(), StrInv()
	LIBMISC.ACT - WaitKC()
	Runtime - StrB()

Description

Displays value bP, considered the starting/default value, and allows value change via spinner controls. The lowest value is limited to bL. The maximum value is limited to bM.

Any byte value is allowed for the limits and default value. The upper limit can be up to 252. This is because the input gadgets use the the 253, 254, and 255 as specific return values that indicate how the gadget was exited. The spinner gadget is an exception in that is a hybrid. It will return those values, and it returns the selected value. Realistically, the foreseen use case is from 0 to 100.



Value: 10

New:

Keys accepted are:

LEFT\+	= Decrease value
RIGHT*	= Increase value
UP\-	= Increase value
DOWN\=	= Decrease value
ESCAPE	= Exits without setting value (returns XESC)
TAB	= Exits without setting value (returns XTAB)
ENTER	= Accepts current value and exits (returns value)

Menus (LIBMENU.ACT)

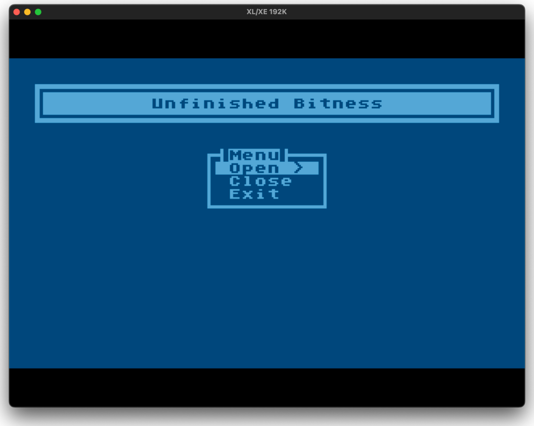
`byte FUNC MenuV(byte bN, x, y, bI, bD, bS char pointer pS)`

Parameters: bN = Window handle number
x = Window column to display menu at
y = Window row to display menu at
bI = Inverse selection on exit flag
    WON = Leave menu selection in inverse video
    WOFF = Return menu selection to normal video
bD = Start item selection number
bS = Menu item width
pS = String containing menu items

Returns: byte = Number of item chosen
    XESC = User ESCaped from menu (no item chosen)
    XTAB = User TABbed from menu (no item chosen)

Requires:
```
DEFINES.ACT
DEFWIN.ACT
LIBWIN.ACT    - WPrint()
LIBMISC.ACT   - WaitKC()
Runtime       - SCopyS()
```

Description
Displays a list of menu items at the x and y coordinates within the window referenced by window handle bN.



The currently selected menu item will be highlighted (displayed in inverse video), while the remaining items will be in normal video.

The menu will have the following navigation key controls:

UP/-	= Move cursor (selection) up
DOWN/=	= Move cursor (selection) down
LEFT/+	= Move cursor (selection) up
RIGHT/*	= Move cursor (selection) down
ENTER	= Accept selected item
ESCAPE	= Abandon selection and return
TAB	= Abandon selection and return

The initially selected item will be the one referenced by bD.

If the selector scrolls past the bottom it will be returned to the top. Likewise if the selector scrolls past the top it will set to the bottom.

If the inverse on exit parameter (bI) is set to WON, the currently highlighted (selected) menu item will remain in inverse video at exit. This is useful if you have sub-menus and want to see the "breadcrumbs" of previous selections.

If the inverse on exit parameter (bI) is set to WOFF, the currently highlighted (selected) menu item will be re-displayed in normal video at exit. This is useful for generating input forms and using MenuV() as a field selector.

The number of the item selected will be returned once a selection is accepted.
If ESCAPE is used to exit the menu, it will return 0 (XESC).
If TAB is used to exit the menu, it will return 99 (XTAB).

Notes

- Version 1.2 introduced a breaking change with two new parameters bI and bD. Parameter order has also changed. Any programs written for previous library versions that use MenuV() will need to be updated before successful compilation and run will occur.

Input/Output (LIBIO.ACT)

card FUNC GetCD(byte bD)

Parameters: bD = Device handle number
Returns: card = Value of card read from device
Requires: Runtime - GetD()

Description
Gets a card value (two bytes) from the device referenced by handle bD. Bytes are read in little endian format (LSB followed by MSB).

The value returned is computed with the following formula:
card = (MSB * 256) + LSB

int FUNC GetID(byte bD)

Parameters: bD = Device handle number
Returns: int = Value (positive or negative) of integer read from device
Requires: LIBIO.ACT - GetCD()

Description
Gets an integer value (two bytes) from the device referenced by handle bD. Bytes are read in little endian format (LSB followed by MSB).

The value returned is computed with the following formula:
card = (MSB * 256) + LSB

It is then checked to see if it is negative, and appropriately assigned if so.

void PROC PutCD(byte bD card cE)

Parameters: bD = Device handle number
cE = Card value to put
Returns: void
Requires: Runtime - PutD()

Description
Puts a card value (two bytes) referenced by cE on the device referenced by handle bD. Bytes are written in little endian format (LSB followed by MSB).

void PROC PutID(byte bD int iV)

Parameters: bD = Device handle number
iV = Integer value to put
Returns: void
Requires: LIBIO.ACT - PutCD()

Description
Puts an integer value (two bytes) referenced by iV on the device referenced by handle bD. Bytes are written in little endian format (LSB followed by MSB).

void PROC EatByteD(byte bD card cB)

Parameters: bD = Device handle number
cB = Number of bytes to eat
Returns: void
Requires: n/a

Description
Reads cB number of bytes from the device referenced by handle bD. Bytes are discarded after being read.

Serial Input/Output (LIBSIO.ACT)

void PROC SIOV(void)

Parameters: n/a
Returns: n/a
Requires: n/a

Description
Call the SIO vector of the operating system at location `$E459`.

It is assumed the SIO DCB (device control block) has been appropriately defined with values before calling `SIOV()`.

The SIO DCB variables are defined as part of this library. Those variables are predefined to point to OS memory locations as defined below:

```
BYTE DDEVIC = $300 - Device bus serial ID
     DUNIT  = $301 - Device unit number
     DCOMND = $302 - Device operation (command) number
     DSTATS = $303 - Device status (device dependent)
     DTIMLO = $306 - Device timeout in seconds (default 31 units or 30 seconds)
     DAUX1  = $30A - Auxillary byte 1 (device dependent)
     DAUX2  = $30B - Auxillary byte 2 (device dependent)
CARD DBUF   = $304 - Data buffer address (2 bytes as LSB/MSB)
     DBYT   = $308 - Data transfer size (2 bytes as LSB/MSB)
```

To define a value into these locations, you can simply set the variable to the value. Example for setting up APETime call:

```
; APETime=Device 69 ($45), Unit 1
; Time command=147 ($93)
; Get 6 byte and store in byte array address of bA
; Timeout just over 15s
DDEVIC=69
DUNIT=1
DCOMND=147
DSTATS=64
DTIMLO=15
DBUF=bA
DBYT=6
```

String Manipulation (LIBSTR.ACT)

void PROC StrAI(char pointer pS)

Parameters:	pS = Pointer to text string
Returns:	void
Requires:	n/a

Description
Converts string referenced by pS from the ATASCII code representation to the internal code representation.

This is generally useful for putting characters or copying text strings directly to screen memory.

void PROC StrIA(char pointer pS)

Parameters:	pS = Pointer to text string
Returns:	void
Requires:	n/a

Description
Converts string referenced by pS from the internal code representation to the ATASCII code representation.

This is the opposite of StrAI().

void PROC StrInv(char pointer pS byte bS)

Parameters:	pS = Pointer to text string bS = Number of bytes to inverse
Returns:	void
Requires:	n/a

Description
Inverses (inverse video) the string referenced by pS up to size bS bytes in length.

`void PROC StrPad(char pointer pS byte bC, bL)`

Parameters: `pS` = Pointer to text string
`bC` = Character to pad string with
`bL` = Length to pad the string to
Returns: `void`
Requires: `Runtime      - SAssign(), SCopy(), SetBlock()`

Description
Pads the string referenced by `pS` with character `bC` up to size `bL` bytes in length. The maximum length is 10 characters.

`void PROC SubStr(char array cB, cE byte bN, bS)`

Parameters: `cB` = Text string to take substring from (source)
`cE` = Text string to place substring into (destination)
`bN` = Starting position of substring in source
`bS` = Number of characters to copy into substr
Returns: `void`
Requires: n/a

Description
Copies a substring of `bS` characters from the string referenced by `cB` starting at position `bN`, and places result in the character string referenced by `cE`.

`void PROC StrTrim(char pointer pS)`

Parameters: `pS` = Pointer to text string
Returns: `void`
Requires: n/a

Description
Removes trailing spaces from the string referenced by `pS`.

DOS (LIBDOS.ACT)

byte FUNC IsSD()

Parameters:	void	
Returns:	byte =	1 = SpartaDOS
		0 = Non-SpartaDOS
Requires:	n/a	

Description
Determines if DOS is SpartaDOS.

void PROC SDx()

Parameters:	void
Returns:	void
Requires:	n/a

Description
Exits program by jumping to DOS through `DOSVEC ($000A)`.

Miscellaneous (LIBMISC.ACT)

byte FUNC IKC2ATA(byte bN)

Parameters: bN = Internal key code
Returns: byte = ATASCII character code
Or unconverted internal code (see Description)
Or KNOMAP (199) for internal codes with no character mapping (see Description)
Requires: DEFINES.ACT

Description
Converts internal key code to ATASCII character code.

Performs conversion for all internal key codes with value less than 192. If the internal code passed in is greater than 191, it is returned unmodified. If the internal code passed in is greater than 127 and does not have a character mapping, KNOMAP (199) is returned. Key code 199 is not bound to any keystroke combination.

void PROC Wait(byte bN)

Parameters: bN = Number of seconds to wait
Returns: void

Description
Waits bN number of seconds.

card FUNC WaitKC()

Parameters: void
Returns: card = key code value of key pressed
Requires: DEFINES.ACT

Description
Waits for any keystroke or console key press. The function does not process functions for transient keys like Inverse or Caps, though it will return the key stroke value.

The keypress is consumed before returning.

card FUNC WaitKCX(byte bI)

Parameters: bI = Flag to execute inverse function or not
1 = Yes
0 = No
Returns: card = key code value of key pressed
Requires: DEFINES.ACT

Description
Waits for any keystroke, function key, or console key press. Function keys include HELP, and F1 through F4. This function will process transient keys Caps and Inverse as well as returning the key stroke value. This means caps-lock will be toggled on and off as the key is pressed.

The transient inverse keystroke will be toggled only if bI is passed as 1.

The keypress is consumed before returning.

This is an expanded version of WaitKC intended for use on XL/XE computers.

byte FUNC WaitYN(byte bE)

Parameters: bE = Flag for display of ? prompt
1 = Display ?
0 = Do not display ?
Returns: byte = 1 = Y or y pressed
0 = N or n pressed
Requires: DEFINES.ACT
Runtime - Put()

Description
Waits for a Y or N keystroke. Upper and lower case letters are accepted. Will optionally display a ‘?’ character at the current virtual window system cursor location if bE is set to 1.

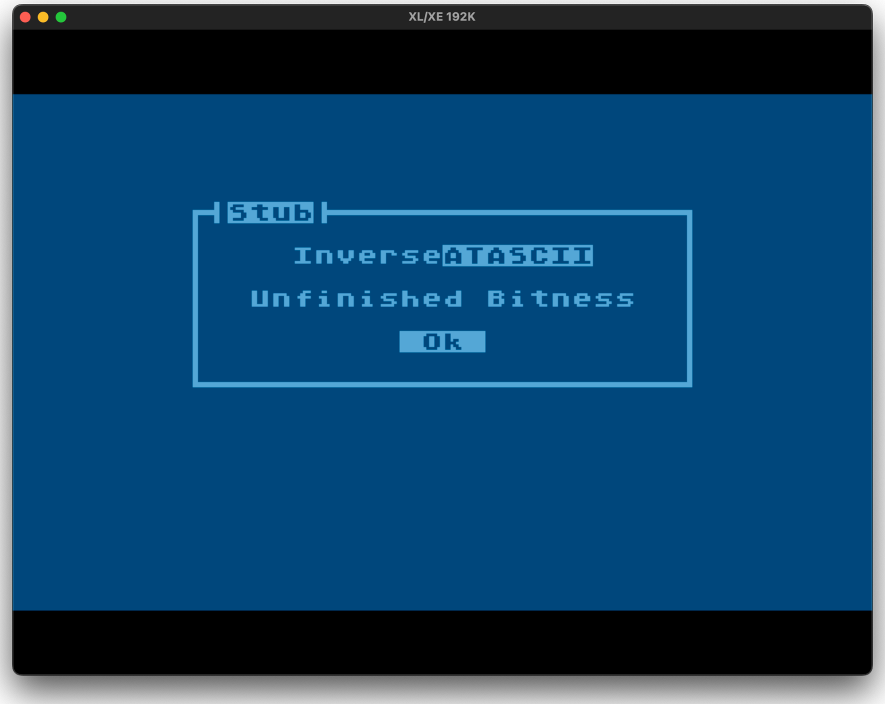
The keypress is consumed before returning.

Usage Examples

Stub Programs

Stub Window

This demonstrates the very basics of the window system. It shows how to include the library and open a window.



Stub

InverseATASCII

Unfinished Bitness

Ok

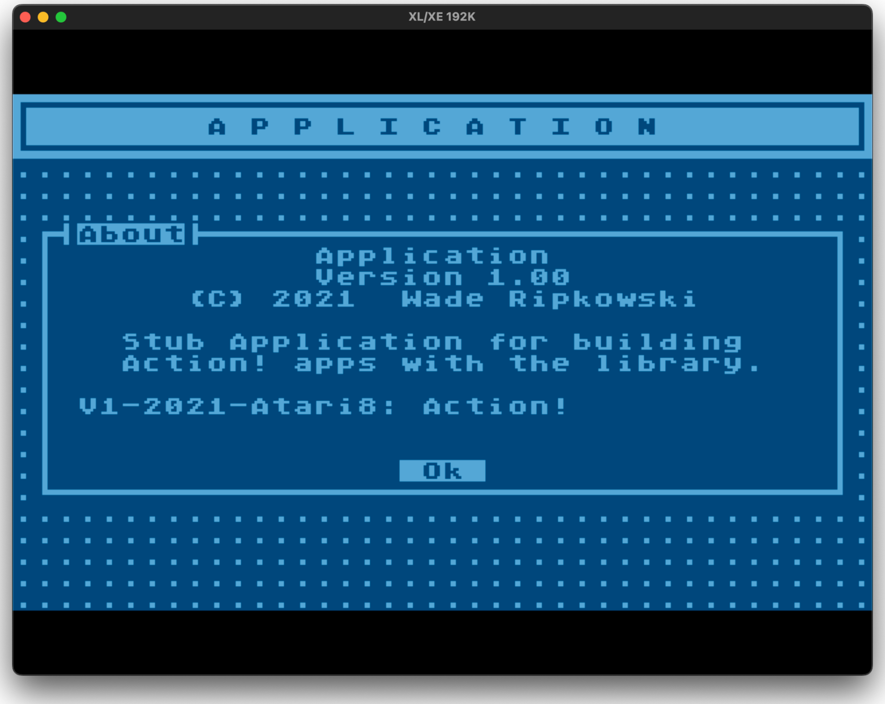
File: STUBWIN.ACT

```
; Program: STUBWIN.ACT
; Author.: Wade Ripkowski
; Date...: 2016.07
; Desc...: Stub Window Program
; License: Creative Commons
;          Attribution-NonCommercial-
;          NoDerivatives
;          4.0 International

; Include library
INCLUDE "D1:DEFINES.ACT"
INCLUDE "D1:DEFWIN.ACT"
INCLUDE "D1:LIBSTR.ACT"
INCLUDE "D1:LIBWIN.ACT"
INCLUDE "D1:LIBMISC.ACT"

; Start
MODULE

PROC Main()
; Window handles
BYTE bW1

; Init Window System
```

```
WInit()

; Open window 1
bW1=WOpen(8,5,24,9,WOFF)
WOrn(bW1,WPTOP,WPLFT,"Stub")
WPrint(bW1,WPCNT,2,"InverseATASCII")
WPrint(bW1,WPCNT,4,"Unfinished Bitness")
WPrint(bW1,WPCNT,6," Ok ")

; Wait for a keystroke or console key
WaitKC()

; Close window 1
WClose(bW1)

RETURN
```

Stub Application Shell

This demonstrates a shell application using the window system. It shows how to include the library and build the foundation of a larger application.



```
         Application
         Version 1.00
   (C) 2021  Wade Ripkowski

 Stub Application for building
 Action! apps with the library.

V1-2021-Atari8: Action!
```

File: STUBAPP.ACT

```
; Program: STUBAPP.ACT
; Author.: Wade Ripkowski
; Date...: 2021.01
; Desc...: Stub Application
; License: Creative Commons
;          Attribution-NonCommercial-
;          NoDerivatives
;          4.0 International

; Include library
INCLUDE "D1:DEFINES.ACT"
INCLUDE "D1:DEFWIN.ACT"
INCLUDE "D1:LIBSTR.ACT"
INCLUDE "D1:LIBWIN.ACT"
INCLUDE "D1:LIBMISC.ACT"
INCLUDE "D1:LIBGADG.ACT"
INCLUDE "D1:LIBMENU.ACT"

; Start
MODULE

; ------------------------------
; Proc: About()
; Desc: About Dialog
; ------------------------------
PROC About()
BYTE bW1

; Show window
bW1=WOpen(1,6,38,14,WOFF)
WOrn(bW1,WPTOP,WPLFT,"About")
WPrint(bW1,WPCNT,1,"Application")
WPrint(bW1,WPCNT,2,"Version 1.00")
WPrint(bW1,WPCNT,3,"(C) 2021  Wade Ripkowski")
WPrint(bW1,WPCNT,5,"Stub Application for building")
WPrint(bW1,WPCNT,6,"Action! apps with the library.")
WPrint(bW1,2,8, "V1-2021-Atari8: Action!")
WPrint(bW1,WPCNT,12," Ok ")

; Wait for key
WaitKC()

; Close window
WClose(bW1)

RETURN


; ------------------------------
; Proc: SubMenu3()
; Desc: Sub menu 3 routine
; ------------------------------
PROC SubMenu3()
BYTE bW1,bCh
CHAR ARRAY cM(37)

; Open window
bW1=WOpen(16,10,14,5,WOFF)
WOrn(bW1,WPTOP,WPCNT,"Sub-Menu 3")

; Build menu
SCopy(cM," Sub-Item 1  Sub-Item 2  Sub-Item 3 ")

; Do until exit
DO
  ; Display menu and get choice
  bCh=MenuV(bW1,1,1,WOFF,1,12,cM)

  ; Process choice
  if bCh=XESC then
    exit

  elseif bCh=1 then
    GAlert(" Sub-Item 1 selected. ")

  elseif bCh=2 then
    GAlert(" Sub-Item 2 selected. ")

  elseif bCh=3 then
    GAlert(" Sub-Item 3 selected. ")
  fi
OD

; Close window
WClose(bW1)

RETURN


; ------------------------------
; Proc: Main()
; Desc: Main routine
; ------------------------------
PROC Main()
BYTE bW1,bW2,bCh
CHAR ARRAY cM(61)

; Init Window System
WInit()

; Set Background
WBack(14)

; Open header window
bW1=WOpen(0,0,40,3,WON)
WPrint(bW1,WPCNT,1,"A P P L I C A T I O N")

; Open menu window
bW2=WOpen(13,7,12,9,WOFF)
WOrn(bW2,WPTOP,WPCNT,"Menu")

; Build menu
SCopy(cM,
" Sub-Menu 1  Sub-Menu 2  Sub-Menu 3  About       Exit       ")

; Do until exit
DO
  ; Display menu and get choice
  bCh=MenuV(bW2,1,2,WOFF,1,12,cM)

  ; Process choice
  if bCh=1 then
    GAlert(" Sub-Menu 1 selected. ")

  elseif bCh=2 then
    GAlert(" Sub-Menu 2 selected. ")

  elseif bCh=3 then
    SubMenu3()

  elseif bCh=4 then
    About()

  elseif bCh=XESC or bCh=5 then
    exit
  fi
OD

; Close windows
WClose(bW2)
WClose(bW1)

RETURN
```

Stub Input Form

This demonstrates an input form using the window system, menu, and gadgets. It shows how to include the library and usage of the input gadgets.

File: STUBFORM.ACT

```
; Program: STUBFORM.ACT
; Author.: Wade Ripkowski
; Date...: 2021.01
; Desc...: Form Input Test
; Notes..: !!! Before Compiling !!!
;          MUST RUN BIGST.ACT 1st!
;          With: bigST='D
;          Then: SET $495=12

; Include library
INCLUDE "D1:DEFINES.ACT"
INCLUDE "D1:DEFWIN.ACT"
INCLUDE "D1:LIBSTR.ACT"
INCLUDE "D1:LIBMISC.ACT"
INCLUDE "D1:LIBWIN.ACT"
INCLUDE "D1:LIBGADG.ACT"
INCLUDE "D1:LIBMENU.ACT"

; Start
MODULE


; -----------------------------------
; Func..: Form()
; Descr.: Demonstation input form
;         using multiple gadgets.
; -----------------------------------
BYTE FUNC Form()
BYTE bR=[FALSE]
BYTE bW1,bM,bA,bB,bC,bD
BYTE bRA,bRB,bRAp,bRBp
BYTE bCha,bChb,bChc,bChap,bChbp,bChcp
CHAR ARRAY cA(41),cB(41),cC(41),cD(41)
CARD ARRAY aB(2)
CARD ARRAY rA(3),rB(3)


; Strings for navigation footer
CHAR ARRAY cF="Nav:↑↓←→▶€↵   "
CHAR ARRAY cI="Nav:←→€↵^cS^cE"
CHAR ARRAY cR="Nav:↑↓←→▶€ ↵  "
CHAR ARRAY cX="Nav:X ▶€↵     "

; Setup buttons
; Element 0 will be seletion 1
aB(0)="[ Ok ]"
aB(1)="[Cancel]"

; Set radio buttons and defaults
rA(0)="One"
rA(1)="Two"
rA(2)="Three"
rB(0)="Choice A"
rB(1)="Choice B"
rB(2)="Choice C"
bRA=1
bRB=1
bRAp=bRA
bRBp=bRB

; Prep strings
SCopy(cA,"-100.00                                 ")
SCopy(cB,"This string has something to edit in it!")
SCopy(cC,"                                        ")
SCopy(cD,"Any String!                             ")

; Set checkbox defaults for previous
bChap=GCOFF
bChbp=GCON
bChcp=GCOFF
```

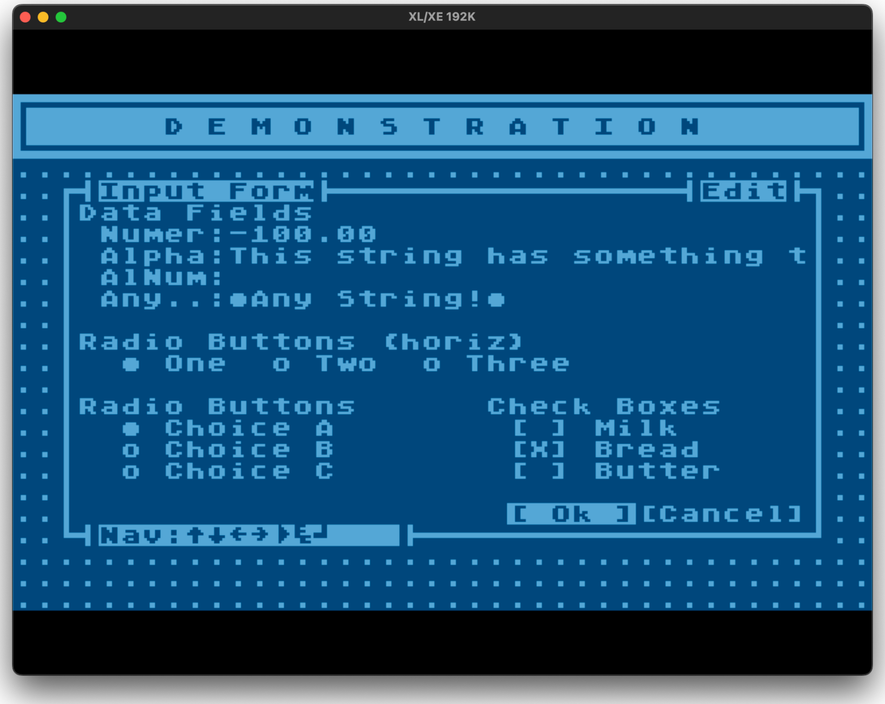
```
; Open window & draw contents
bW1=WOpen(2,2,36,17,WOFF)
WOrn(bW1,WPTOP,WPLFT,"Input Form")
WOrn(bW1,WPTOP,WPRGT,"Edit")
WOrn(bW1,WPBOT,WPLFT,cF)

WPrint(bW1,1,1,"Data Fields")
WPrint(bW1,2,2,"Numer:")
WPrint(bW1,2,3,"Alpha:")
WPrint(bW1,2,4,"AlNum:")
WPrint(bW1,2,5,"Any..:")

WPrint(bW1,1,7,"Radio Buttons (horiz)")
GRadio(bW1,2,8,GHORZ,GDISP,bRAp,3,rA)

WPrint(bW1,1,10,"Radio Buttons")
GRadio(bW1,2,11,GVERT,GDISP,bRBp,3,rB)

WPrint(bW1,20,10,"Check Boxes")
WPrint(bW1,25,11,"Milk")
WPrint(bW1,25,12,"Bread")
WPrint(bW1,25,13,"Butter")
GCheck(bW1,21,11,GDISP,bChap)
GCheck(bW1,21,12,GDISP,bChbp)
GCheck(bW1,21,13,GDISP,bChcp)

GButton(bW1,21,15,GDISP,2,aB)

; Display fields as is
WPrint(bW1,8,2,cA)
WPrint(bW1,8,3,cB)
WPrint(bW1,8,4,cC)
WPrint(bW1,8,5,cD)

; Loop until form accepted or cancelled
DO
  ; Set initial menu selection
  bM=1

  ; Loop until user ESCapes or TABs out
  DO
    ; Cycle through fields
    bM=MenuV(bW1,2,2,WOFF,bM,5,"NumerAlphaAlNumAny..")

    ; Edit the chosen field
    if bM=1 then
      WOrn(bW1,WPBOT,WPLFT,cI)
      bA=GInput(bW1,8,2,GNUMER,27,cA)
      WOrn(bW1,WPBOT,WPLFT,cF)

    elseif bM=2 then
      WOrn(bW1,WPBOT,WPLFT,cI)
      bB=GInput(bW1,8,3,GALPHA,27,cB)
      WOrn(bW1,WPBOT,WPLFT,cF)

    elseif bM=3 then
      WOrn(bW1,WPBOT,WPLFT,cI)
      bC=GInput(bW1,8,4,GALNUM,27,cC)
      WOrn(bW1,WPBOT,WPLFT,cF)

    elseif bM=4 then
      WOrn(bW1,WPBOT,WPLFT,cI)
      bD=GInput(bW1,8,5,GANY,27,cD)
      WOrn(bW1,WPBOT,WPLFT,cF)
    fi
  UNTIL bM=XESC or bM=XTAB
  OD

  ; Display radio buttons - horizontal
  WOrn(bW1,WPBOT,WPLFT,cR)
  bRA=GRadio(bW1,2,8,GHORZ,GEDIT,bRAp,3,rA)
  if bRA#XESC and bRA#XTAB then
    bRAp=bRA
  fi
  GRadio(bW1,2,8,GHORZ,GDISP,bRAp,3,rA)

  ; Display radio buttons - veritcal
  bRB=GRadio(bW1,2,11,GVERT,GEDIT,bRBp,3,rB)
  if bRB#XESC and bRB#XTAB then
    bRBp=bRB
  fi
  GRadio(bW1,2,11,GVERT,GDISP,bRBp,3,rB)
  WOrn(bW1,WPBOT,WPLFT,cF)

  ; Check boxes, set footer
  WOrn(bW1,WPBOT,WPLFT,cX)

  ; Stay on this check until ESC, TAB, or SET
  DO
    ; Display button and get choice
    bCha=GCheck(bW1,21,11,GEDIT,bChap)

    ; If ESC or TAB, exit loop
    if bCha=XESC or bCha=XTAB then
      exit
    else
      ; Else, assign return to previous
      bChap=bCha
    fi
  OD

  ; Stay on this check until ESC, TAB, or SET
  DO
    bChb=GCheck(bW1,21,12,GEDIT,bChbp)

    ; If ESC or TAB, exit loop
    if bChb=XESC or bChb=XTAB then
      exit
    else
      ; Else, assign return to previous
      bChbp=bChb
    fi
  OD

  ; Stay on this check until ESC, TAB, or SET
  DO
    bChc=GCheck(bW1,21,13,GEDIT,bChcp)

    ; If ESC or TAB, exit loop
    if bChc=XESC or bChc=XTAB then
      exit
    else
      ; Else, assign return to previous
      bChcp=bChc
    fi
  OD

  ; Restore footer
  WOrn(bW1,WPBOT,WPLFT,cF)

  ; If ESC out of fields, dont do buttons
  if bM#XESC then
    ; Prompt for form acceptance
    bM=GButton(bW1,21,15,1,2,aB)

    ; Redraw buttons
    GButton(bW1,21,15,GDISP,2,aB)
  fi
UNTIL bM#XTAB
OD

; Do something with data if accepted, set true exit
if bM=1 then
  bR=TRUE
  GAlert("Doing something with entered data...")
fi

; Close window
WClose(bW1)

RETURN(bR)


; ----------------------------------
; Main routine
; ----------------------------------
PROC Main()
BYTE bW1,bR

; Init Window System
WInit()

; Call form
bR=Form()

; Check form return status
if bR=TRUE then
  GAlert("Returned TRUE (edited)")
else
  GAlert("Returned FALSE (escaped)")
```

```
fi
RETURN
```

Demo Program

Demonstration Application

This demonstrates a fully functioning application using the window system, and several gadgets. It shows how to include the library and build the foundation of a larger application.



```
D E M O N S T R A T I O N

 Input Form                           Edit
Data Fields
  Numer:-100.00
  Alpha:This string has something t
  AlNum:
  Any..:●Any String!●

Radio Buttons (horiz)
   ● One   o Two   o Three

Radio Buttons                Check Boxes
   ● Choice A                  [ ] Milk
   o Choice B                  [X] Bread
   o Choice C                  [ ] Butter

                                 [ Ok ][Cancel]
 Nav:↑↓←→⏎
```

File: DEMOAPP.ACT

```
; Program: APPDEMO.ACT
; Author.: Wade Ripkowski
; Date...: 2021.01
; Desc...: Demo Application
; License: Creative Commons
;          Attribution-NonCommercial-
;          NoDerivatives
;          4.0 International
; Notes..: MUST run BIGST.ACT 1st w/
;          bigST='D
;          Then: SET $495=12

; Include library
INCLUDE "D1:DEFINES.ACT"
INCLUDE "D1:DEFWIN.ACT"
INCLUDE "D1:LIBSTR.ACT"
INCLUDE "D1:LIBWIN.ACT"
INCLUDE "D1:LIBMISC.ACT"
INCLUDE "D1:LIBGADG.ACT"
INCLUDE "D1:LIBMENU.ACT"

; Start
MODULE

; -----------------------------------
; Func..: FormInput()
; Desc..: Demo use of input gadgets
; Return: TRUE if accepted, or FALSE
; -----------------------------------
```

```
BYTE FUNC FormInput()
BYTE bR=[FALSE]
BYTE bW1,bM,bA,bB,bC,bD
BYTE bRA,bRB,bRAp,bRBp
BYTE bCha,bChb,bChc,bChap,bChbp,bChcp
CHAR ARRAY cA(41),cB(41),cC(41),cD(41)
CARD ARRAY aB(2)
CARD ARRAY rA(3),rB(3)


; Strings for navigation footer
CHAR ARRAY cF="Nav:↑↓←→▶␛↵   "
CHAR ARRAY cI="Nav:←→␛↵^cS^cE"
CHAR ARRAY cR="Nav:↑↓←→▶␛ ↵  "
CHAR ARRAY cX="Nav:X ▶␛↵     "


; Setup buttons
; Element 0 will be seletion 1
aB(0)="[ Ok ]"
aB(1)="[Cancel]"

; Set radio buttons and defaults
rA(0)="One"
rA(1)="Two"
rA(2)="Three"
rB(0)="Choice A"
rB(1)="Choice B"
rB(2)="Choice C"
bRA=1
bRB=1
bRAp=bRA
bRBp=bRB

; Prep strings
SCopy(cA,"-100.00                                 ")
SCopy(cB,"This string has something to edit in it!")
SCopy(cC,"                                        ")
SCopy(cD,"Any String!                           ")

; Set checkbox defaults for previous
bChap=GCOFF
bChbp=GCON
bChcp=GCOFF

; Open window & draw contents
bW1=WOpen(2,4,36,17,WOFF)
WOrn(bW1,WPTOP,WPLFT,"Input Form")
WOrn(bW1,WPTOP,WPRGT,"Edit")
WOrn(bW1,WPBOT,WPLFT,cF)

WPrint(bW1,1,1,"Data Fields")
WPrint(bW1,2,2,"Numer:")
WPrint(bW1,2,3,"Alpha:")
WPrint(bW1,2,4,"AlNum:")
WPrint(bW1,2,5,"Any..:")

WPrint(bW1,1,7,"Radio Buttons (horiz)")
GRadio(bW1,2,8,GHORZ,GDISP,bRAp,3,rA)

WPrint(bW1,1,10,"Radio Buttons")
GRadio(bW1,2,11,GVERT,GDISP,bRBp,3,rB)

WPrint(bW1,20,10,"Check Boxes")
WPrint(bW1,25,11,"Milk")
WPrint(bW1,25,12,"Bread")
WPrint(bW1,25,13,"Butter")
GCheck(bW1,21,11,GDISP,bChap)
GCheck(bW1,21,12,GDISP,bChbp)
GCheck(bW1,21,13,GDISP,bChcp)

GButton(bW1,21,15,GDISP,2,aB)

; Display fields as is
WPrint(bW1,8,2,cA)
WPrint(bW1,8,3,cB)
WPrint(bW1,8,4,cC)
WPrint(bW1,8,5,cD)

; Loop until form accepted or cancelled
DO
  ; Set initial menu selection
  bM=1
```

```
; Loop until user ESCapes or TABs out
DO
  ; Cycle through fields
  bM=MenuV(bW1,2,2,WOFF,bM,5,"NumerAlphaAlNumAny..")

  ; Edit the chosen field
  if bM=1 then
    WOrn(bW1,WPBOT,WPLFT,cI)
    bA=GInput(bW1,8,2,GNUMER,27,cA)
    WOrn(bW1,WPBOT,WPLFT,cF)

  elseif bM=2 then
    WOrn(bW1,WPBOT,WPLFT,cI)
    bB=GInput(bW1,8,3,GALPHA,27,cB)
    WOrn(bW1,WPBOT,WPLFT,cF)

  elseif bM=3 then
    WOrn(bW1,WPBOT,WPLFT,cI)
    bC=GInput(bW1,8,4,GALNUM,27,cC)
    WOrn(bW1,WPBOT,WPLFT,cF)

  elseif bM=4 then
    WOrn(bW1,WPBOT,WPLFT,cI)
    bD=GInput(bW1,8,5,GANY,27,cD)
    WOrn(bW1,WPBOT,WPLFT,cF)
  fi
UNTIL bM=XESC or bM=XTAB
OD

; Display radio buttons - horizontal
WOrn(bW1,WPBOT,WPLFT,cR)
bRA=GRadio(bW1,2,8,GHORZ,GEDIT,bRAp,3,rA)
if bRA#XESC and bRA#XTAB then
  bRAp=bRA
fi
GRadio(bW1,2,8,GHORZ,GDISP,bRAp,3,rA)

; Display radio buttons - veritcal
bRB=GRadio(bW1,2,11,GVERT,GEDIT,bRBp,3,rB)
if bRB#XESC and bRB#XTAB then
  bRBp=bRB
fi
GRadio(bW1,2,11,GVERT,GDISP,bRBp,3,rB)
WOrn(bW1,WPBOT,WPLFT,cF)

; Check boxes, set footer
WOrn(bW1,WPBOT,WPLFT,cX)

; Stay on this check until ESC, TAB, or SET
DO
  ; Display button and get choice
  bCha=GCheck(bW1,21,11,GEDIT,bChap)

  ; If ESC or TAB, exit loop
  if bCha=XESC or bCha=XTAB then
    exit
  else
    ; Else, assign return to previous
    bChap=bCha
  fi
OD

; Stay on this check until ESC, TAB, or SET
DO
  bChb=GCheck(bW1,21,12,GEDIT,bChbp)

  ; If ESC or TAB, exit loop
  if bChb=XESC or bChb=XTAB then
    exit
  else
    ; Else, assign return to previous
    bChbp=bChb
  fi
OD

; Stay on this check until ESC, TAB, or SET
DO
  bChc=GCheck(bW1,21,13,GEDIT,bChcp)

  ; If ESC or TAB, exit loop
  if bChc=XESC or bChc=XTAB then
    exit
  else
    ; Else, assign return to previous
    bChcp=bChc
    fi
  OD

  ; Restore footer
  WOrn(bW1,WPBOT,WPLFT,cF)

  ; If ESC out of fields, dont do buttons
  if bM#XESC then
    ; Prompt for form acceptance
    bM=GButton(bW1,21,15,1,2,aB)

    ; Redraw buttons
    GButton(bW1,21,15,GDISP,2,aB)
  fi
UNTIL bM#XTAB
OD

; Do something with data if accepted, set true exit
if bM=1 then
  bR=TRUE
  GAlert("Doing something with entered data...")
fi

; Close window
WClose(bW1)

RETURN(bR)



; ------------------------------------
; Proc..: ProgTest()
; Descr.: Demos window status and
;         progress bar.
; ------------------------------------
PROC ProgTest()
BYTE bW1,bW2,bL,bS
INT iV

; Open status window
bW1=WOpen(9,2,20,14,WOFF)
WOrn(bW1,WPTOP,WPLFT,"Status")
WPrint(bW1,1,1,"Window Status")
WPrint(bW1,1,2,"------ ------")

; Open progress bar window
bW2=WOpen(7,18,24,4,WOFF)
WPrint(bW2,2,1,"Progress:")

; Display initial progress bar
GProg(bW2,2,2,0)

; Loop through each window handle
for bL=0 to 9
DO
  ; Get the status
  bS=WStat(bL)

  ; Print the window handle #
  WPos(bW1,6,3+bL)
  WPut(bW1,bL+48)

  ; Print the handle status
  if bS=WON then
    WPrint(bW1,8,3+bL,"Used")
  else
    WPrint(bW1,8,3+bL,"Free")
  fi

  ; Update progress bar
  iV=((bL+1) MOD 10)*10
  if iV=0 then
    iV=100
  fi
  GProg(bW2,2,2,iV)

  ; Wait 1 second
  Wait(1)
OD

GAlert(" Press a key to continue. ")

; Close windows
WClose(bW2)
WClose(bW1)
```

```
RETURN


; -----------------------------------
; Proc: About()
; Desc: About Dialog
; -----------------------------------
PROC About()
BYTE bW1

; Show window
bW1=WOpen(1,6,38,14,WOFF)
WOrn(bW1,WPTOP,WPLFT,"About")
WPrint(bW1,WPCNT,1,"Demo Application")
WPrint(bW1,WPCNT,2,"Version 1.00")
WPrint(bW1,WPCNT,3,"(C) 2021  Wade Ripkowski")
WPrint(bW1,WPCNT,5,"Application to demonstrate")
WPrint(bW1,WPCNT,6,"the Action! library.")
WPrint(bW1,2,8, "V1-2021-Atari8: Action!")
WPrint(bW1,WPCNT,12," Ok ")

; Wait for key
WaitKC()

; Close window
WClose(bW1)

RETURN


; -----------------------------------
; Proc: SubMenu()
; Desc: Sub menu routine
; -----------------------------------
PROC SubMenu()
BYTE bW1,bCh
CHAR ARRAY cM(37)

; Open window
bW1=WOpen(16,10,14,5,WOFF)
WOrn(bW1,WPTOP,WPLFT,"Sub-Menu")

; Build menu
SCopy(cM," Sub-Item 1  Sub-Item 2  Sub-Item 3 ")

; Do until exit
DO
  ; Display menu and get choice
  bCh=MenuV(bW1,1,1,WOFF,1,12,cM)

  ; Process choice
  if bCh=XESC then
    exit

  elseif bCh=1 then
    GAlert(" Sub-Item 1 selected. ")

  elseif bCh=2 then
    GAlert(" Sub-Item 2 selected. ")

  elseif bCh=3 then
    GAlert(" Sub-Item 3 selected. ")
  fi
OD

; Close window
WClose(bW1)

RETURN


; -----------------------------------
; Proc: Main()
; Desc: Main routine
; -----------------------------------
PROC Main()
BYTE bW1,bW2,bCh
CHAR ARRAY cM(71)

; Init Window System
WInit()

; Set Background
WBack(14)
```

```
; Open header window
bW1=WOpen(0,0,40,3,WON)
WPrint(bW1,WPCNT,1,"D E M O N S T R A T I O N")

; Open menu window
bW2=WOpen(12,7,16,9,WOFF)
WOrn(bW2,WPTOP,WPLFT,"Menu")

; Build menu
SCopy(cM," Input Form    Progress Bar  Sub-Menu      About
Exit         ")

; Do until exit
DO
  ; Display menu and get choice
  bCh=MenuV(bW2,1,2,WOFF,1,14,cM)

  ; Process choice
  if bCh=1 then
    FormInput()

  elseif bCh=2 then
    ProgTest()

  elseif bCh=3 then
    SubMenu()

  elseif bCh=4 then
    About()

  elseif bCh=XESC or bCh=5 then
    exit
  fi
OD

; Close windows
WClose(bW2)
WClose(bW1)

RETURN
```