

# Atari 8 Bit C Library Reference

*Copyright © 2022-2023 All Rights Reserved*

# C

# Atari 8 Bit C Library Reference

Library Version 1.20 (2023.08.26)

Reference Revision A

**Wade Ripkowski**  
**[inverseatascii@icloud.com](mailto:inverseatascii@icloud.com)**

*Copyright © 2022-2023 All Rights Reserved*

# Table of Contents

<b>Overview</b>	<b>1</b>
<b>Local Variables Warning</b>	<b>2</b>
<b>Code Management</b>	<b>3</b>
<b>Size</b>	<b>3</b>
<b>Compilation</b>	<b>3</b>
<b>Organization</b>	<b>4</b>
<b>Requirements</b>	<b>5</b>
<b>C Standard Library Dependency</b>	<b>5</b>
<b>Symbol Table</b>	<b>5</b>
<b>File Reference</b>	<b>6</b>
A8DEFINES.H	6
A8DEFAPE.H	6
A8DEFFUJI.H	6
A8DEFFUJIB64.H	6
A8DEFFUJIHASH.H	6
A8DEFSIO.H	6
A8DEFWIN.H	7
A8LIBAPE.C	7
A8LIBFUJI.C	7
A8LIBFUJIB64.C	7
A8LIBFUJIHASH.C	7
A8LIBGADG.C	7
A8LIBMENU.C	7
A8LIBMISC.C	7
A8LIBSTR.C	8
A8LIBWIN.C	8
<b>API Reference</b>	<b>9</b>
<b>Window System (A8LIBWIN.C)</b>	<b>9</b>
void WBack(byte bN)	9

byte WClose(byte bN)	9
byte WClr(byte bN)	10
byte WDiv(byte bN, byte y, byte bD)	10
void WInit(void)	11
byte WOpen(byte x, byte y, byte w, byte h, byte bT)	12
byte WOrn(byte bN, byte bT, byte bL, unsigned char *pS)	13
byte WPos(byte bN, byte x, byte y)	14
byte WPrint(byte bN, byte x, byte y, byte bI, unsigned char *pS)	15
byte WPut(byte bN, byte x)	16
byte WStat(byte bN)	16
<b>Gadgets (A8LIBGADG.C)</b>	<b>17</b>
void GAlert(unsigned char *pS)	17
byte GButton(byte bN, byte x, byte y, byte bD, byte bS unsigned char **pA)	18
byte GCheck(byte bN, byte x, byte y, byte bI, byte bD)	19
byte GInput(byte bN, byte x, byte y, byte bT, byte bS, unsigned char *pS)	20
void GProg(byte bN, byte x, byte y, byte bS)	22
byte GRadio(byte bN, byte x, byte y, byte bD, byte bE, byte bI, byte bS, unsigned char **pS)	23
byte GSpin(byte bN, byte x, byte y, byte bL, byte bM, byte bI, byte bE)	25
<b>Menus (A8LIBMENU.C)</b>	<b>26</b>
byte MenuV(byte bN, byte x, byte y, byte bI, byte bS, byte bC, unsigned char **pS)	26
<b>String Manipulation (A8LIBSTR.C)</b>	<b>28</b>
void StrAI(unsigned char *pS, byte bS)	28
void StrIA(unsigned char *pS, byte bS)	28
void StrInv(unsigned char *pS, byte bS)	28
<b>APE Atari Peripheral Emulator (A8LIBAPE.C)</b>	<b>29</b>
void ApeTimeG(char *cB)	29
<b>FujiNet (A8LIBFUJI.C)</b>	<b>30</b>
unsigned short FNCheck(void)	30
unsigned char FNClose(void)	30
unsigned char FNGConfig(td_fncfg *sC)	31
unsigned char FNOpen(char *cB)	32
unsigned char FNRead(unsigned short iS, unsigned char *cB)	32
unsigned char FNStatus(void)	33
void FNTrans(unsigned char bM, unsigned char bD)	33
unsigned char FNWrite(unsigned char *cB)	34
<b>FujiNet Base64 (A8LIBFUJIB64.C)</b>	<b>35</b>

unsigned char FNB64Cmp(char cT)	35
unsigned char FNB64Inp(char cT, char *cB, unsigned short iS)	36
unsigned char FNB64Len(char cT, unsigned long *iS)	36
unsigned char FNB64Out(char cT, char *cB, unsigned short iS)	37
<b>FujiNet Hash (A8LIBFUJHASH.C)</b>	<b>38</b>
unsigned char FNHashCmp(byte bT)	38
unsigned char FNHashInp(char *cB, unsigned short iS)	39
unsigned char FNHashOut(byte bT, byte bH, char *cB)	40
<b>Miscellaneous (A8LIBMISC.ACT)</b>	<b>41</b>
byte IKC2ATA(byte bN)	41
byte WaitYN(byte bD)	41
word WaitKCX(byte bI)	42
<b>Usage Examples</b>	<b>43</b>
<b>Stub Programs</b>	<b>43</b>
Stub Window	43
Stub Application Shell	45
<b>Demo Programs</b>	<b>49</b>
Demonstration Application	49
Demonstration APE Time Application	58
Demonstration FujiNet Application	60

# Overview

The library described herein is designed for use with the C programming language, specifically the CC65 dialect with the intended target platform the Atari 8 bit home computer.

The library was initially written in Action! in 2015 and included only the base text windowing system, sort of a TUI (text user interface). Over time it was expanded to include general purpose routines, and gadgets (which are windowing system add-ons). The C version of the library is a conversion of the Action! library done in 2022. It includes all the windows and gadgets of the Action! library, and utilizes direct screen memory I/O like the Action! library. There are a few functions that are either a) not yet converted; or b) not needed due to C having a similar function already. In 2023, the APE, FujiNet, and SIO functions were added.

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 / SIO**

- The input/output routines written in Action! were not converted because C has routines to do the same thing. There is, however, an SIO function.

- **String Manipulation**

- Functions to aid with string manipulation and character conversion.

- **DOS Functions**

- The DOS routines written in Action! were not converted because the CC65 Atari library has routines to do the same thing.

- **APE (Atari Peripheral Emulator)**

- Get time function to retrieve time from attached APE devices real time clock (RTC). Compatible with devices such as FujiNet, or other expansions or SIO2PC/USB software that provide an APE compatible clock.

- **FujiNet**

- Base FujiNet functions and functions for encode and decoding text in Base64 format, as well as functions for hashing text with SHA1, SHA256, and SHA512 algorithms.

- **Miscellaneous**

- Helper functions that don't fall into any particular category, including waiting with and without keystrokes. The Action! Wait() function was not converted because CC65 provides sleep() which does the same thing.

References in this documentation that refer to `void` are meant to represent the C `void` data type.

References in this documentation that refer to `byte` are meant to represent the C `unsigned char` data type.

References in this documentation that refer to `word` are meant to represent the C `unsigned int` data type.

## Local Variables Warning

When programming functions with a large number of variables, or a few variables of a large size, you may run into compilation errors that state there are “Too many local variables”. This is because local variables are normally placed on the stack which has room for 256 bytes.

You can overcome this by using the compiler command line option “-static-locals” which will move local variables to the data segment and off the stack. However doing it via the CLI applies to all the code.

The better way to overcome this is by using compiler directive `#pragma static-locals` before and after the function in the source code. This will isolate moving the locals to the data segment to just that function.

You can see an example of the usage in the libraries example program `appdemo.c` surrounding the `FormInput()` function.

Example usage:

```
#pragma static-locals(push, on)
byte FormInput(void) {
    ...
}
#pragma static-locals(pop)
```

Push will change the setting and save the current setting. Pop restores the saved setting.

# Code Management

## Size

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). This documentation will help identify the dependencies.

One you have identified all of the library functions (and their dependents) used by your application, you will then 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.

## Compilation

In general when compiling a project with CC65, the easiest method is to invoke the compiler and linker together with the CL command (**C**ompile and **L**ink). This may not suit a complex project which could require the use of make.

Typically the call will be:

```
cl -v -O -t atari program.c -o program.xex
```

Parameters:

-v means verbose

-O tells CC65 to optimize. This can reduce xex file size by a few K. Every little bit, right? I recommend not using -O until the program has been fully tested and validated working.

-t tells CC65 which platform to target in regards to memory allocation, load and run locations, etc. I use "atari". You can also use "atarixl" if needed. See the CC65 website for an overview of the differences.

program.c is the name of the program source file.

-o tells CC65 the name to use for the executable.



## Organization

You can pull the library components in as needed, though there is some loose required order due to the library using parts of itself.

First you will want to pull in any CC65 headers that are needed. The following are the base ones used when using all parts of the C Library:

```
#include <stdio.h>
#include <conio.h>
#include <unistd.h>
#include <string.h>
#include <atari.h>
```

“peekpoke.h” is needed as well, but is pulled in by the library sources that need it.

Then you will want to pull in the C Library headers that are needed in this order. All of the library sources require “a8defines.h”. Only “a8libwin.c” requires “a8defwin.h”. It won’t hurt to include these more than once as they establish a definition they have been loaded and only load if the definition is not present.

```
#include "a8defines.h"
#include "a8defwin.h"
```

Last you will pull in the the C Library sources that are needed in this order:

```
#include "a8defsio.h"
#include "a8defape.h"
#include "a8libmisc.c"
#include "a8libstr.c"
#include "a8libwin.c"
#include "a8libgadg.c"
#include "a8libmenu.c"
#include "a8libfuji.c"
#include "a8libfujib64.c"
#include "a8libfujihash.c"
#include "a8libape.c"
```

# Requirements

## C Standard Library Dependency

This library depends upon some of the C standard library functions. Library routines will list their standard function dependencies in the API reference which follows in this documentation.

Routines from the C standard library that are needed:

clrscr()  
cputc()  
gotoxy()  
memcpy()  
memset()  
POKE()  
PEEK()  
PEEKW()  
sprintf()  
strcpy()  
strncpy()  
strlen()

## 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 conserve space.

bC	bT	sC
bD	bU	h
bE	bW	w
bF	bZ	x
bH	cB	xp
bI	cL	y
bK	cR	yp
bL	cS	
bM	cT	baW
bN	iL	baWM
bP	iS	cpWn
bR	pA	vCur
bS	pS	

# File Reference

---

## A8DEFINES.H

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.

---

## A8DEFAPE.H

Atari Peripheral Emulator definitions used by the APE portion of the library.

---

## A8DEFFUJI.H

FujiNet definitions used by the core FujiNet library functions.

This must be included before A8LIBFUJI.C, and before any other FujiNet related files.

---

## A8DEFFUJIB64.H

FujiNet definitions used by the FujiNet Base64 library functions.

This must be included before A8LIBFUJIB64.C.

---

## A8DEFFUJIHASH.H

FujiNet definitions used by the FujiNet Hash (SHA1, SHA256, and SHA512) library functions.

This must be included before A8LIBFUJIBHASH.C.

---

## A8DEFSIO.H

Definitions used by the direct SIO function.

This should always be included BEFORE the APE include file (A8DEFAPE.H), BEFORE the FujiNet include files (A8DEFFUJI.H, A8DEFFUJIB64.H, A8DEFFUJIHASH.H), and before the FujiNet C files (A8DEFFUJI.C, A8DEFFUJIB64.C, A8DEFFUJIHASH.C).

---

## A8DEFWIN.H

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 A8DEFINES.H, and BEFORE A8LIBWIN.C.

---

## A8LIBAPE.C

Function to retrieve time from the APE device real time clock (RTC).

---

## A8LIBFUJI.C

Collection of functions to communicate with the FujiNet. This is the core function set. The FujiNet unit number is assumed to be 1.

---

## A8LIBFUJIB64.C

Collection of functions to utilize the FujiNet's Base64 encoding and decoding functions.

---

## A8LIBFUJIHASH.C

Collection of functions to utilize the FujiNet's Hashing functions. These include SHA1, SHA256, and SHA512 algorithms. FujiNet has a provision for MD5, but it was not implemented at the time of this release.

---

## A8LIBGADG.C

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

When using these routines, A8LIBWIN.C MUST be included before.

---

## A8LIBMENU.C

Collection of menu routines which simplifies program navigation.

When using these routines, A8LIBWIN.C MUST be included before.

---

## A8LIBMISC.C

Collection of routines that don't fall into the other categories, such as waiting for key.

---

## A8LIBSTR.C

Collection of string manipulation routines that augment the C standard library routines which are specific to the Atari platform.

---

## A8LIBWIN.C

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

When using these routines, A8DEFINES.H, A8DEFWIN.H, and A8LIBSTR.C MUST be included before.

# API Reference

## Window System (A8LIBWIN.C)

---

### void WBack(byte bN)

Parameters: bN = Internal code of character

Returns: void

Requires: A8DEFWIN.H  
Standard - memset()

#### 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 WClose(byte bN)

Parameters: bN = Window handle number

Returns: byte = Success status  
0 = Successful  
WENOPN = Window not open (default)

Requires: A8DEFINES.H  
A8DEFWIN.H  
Standard - PEEKW(), memcpy(), memset()

#### 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 WClr(byte bN)

Parameters: bN = Window handle number  
Returns: byte = Success status  
          0 = Successful  
          WENOPN = Window not open (default)  
Requires: A8DEFINES.H  
          A8DEFWIN.H  
          A8LIBSTR.C - StrInv()  
Runtime - PEEKW(), memset(), memcpy()

### Description

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

---

## byte WDiv(byte bN, byte y, byte 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 = Successful  
          WENOPN = Window not open (default)  
Requires: A8DEFINES.H  
          A8DEFWIN.H  
          Standard - PEEKW(), memset(), memcpy()

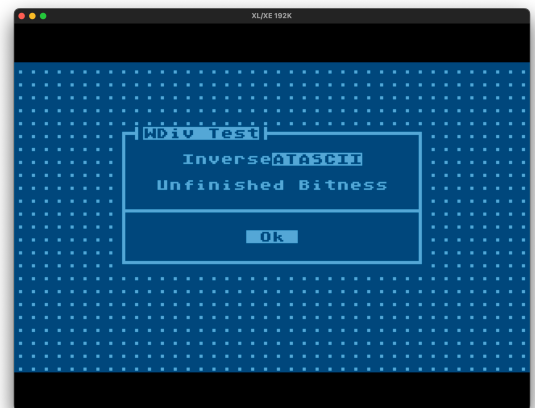
### Description

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.



---

## void WInit(void)

Parameters: void

Returns: void

Requires: A8DEFINES.H

A8DEFWIN.H

Standard - POKE(), clrscr(), gotoxy(), memset()

### 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 A8DEFWIN.H.

---



---

## byte WOpen(byte x, byte y, byte w, byte h, byte 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-

WENONE = No window handles available

Requires: A8DEFINES.H

A8DEFWIN.H

Standard - PEEKW(), memset(), memcpy()

### 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 WOrn(byte bN, byte bT, byte bL, unsigned char *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` = Successful  
    `WENOPN` = Window not open (default)

Requires: `A8DEFINES.H`  
`A8DEFWIN.H`  
`A8LIBSTR.C` - `StrAI()`, `StrInv()`  
Standard - `PEEKW()`, `memcpy()`, `sprintf()`, `strlen()`

### **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 WPos(byte bN, byte x, byte y)

Parameters: bN = Window handle number  
              -or-  
              WPABS = Absolute screen position  
              x = Window column to move cursor to  
              y = Window row to move cursor to

Returns: byte = Success status  
              0 = Successful  
              WENOPN = Window not open

Requires: A8DEFINES.H  
          A8DEFWIN.H  
          Standard - gotoxy()

### 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 WPrint(byte bN, byte x, byte y, byte bI, unsigned char \*pS)

Parameters: bN = Window handle number  
x = Window column to print text  
y = Window row to print text  
bI = Inverse flag  
    WON for inverse  
    WOFF for normal  
pS = Pointer to character string of text to print  
    Maximum size is 38 characters!

Returns: byte = Success status  
    0 = Successful  
    WENOPN = Window not open (default)

Requires: A8DEFINES.H  
    A8DEFWIN.H  
    A8LIBSTR.C     - StrAI(), StrInv()  
    Standard       - PEEKW(), memcpy(), strcpy(), strlen()

### **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. You can force inverse video text by specifying WON in the bI parameter, otherwise use WOFF.

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 WPut(byte bN, byte x)

Parameters: bN = Window handle number  
x = Character to put

Returns: byte = Success status  
0 = Successful  
WENOPN = Window not open

Requires: A8DEFINES.H  
A8DEFWIN.H  
Standard - cputc()

### 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 WStat(byte bN)

Parameters: bN = Window handle number

Returns: byte = Window status  
WON = In use (window ON)  
WOFF = Not in use (window OFF)

Requires: A8DEFWIN.H

### Description

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

## Gadgets (A8LIBGADG.C)

---

`void GAlert(unsigned char *pS)`

Parameters: pS = Pointer to character string to display

Maximum size is 38 characters!

Returns: void

Requires: A8DEFINES.H

A8DEFWIN.H

A8LIBWIN.C - WOpen(), WOrn(), WPrint(), WClose()

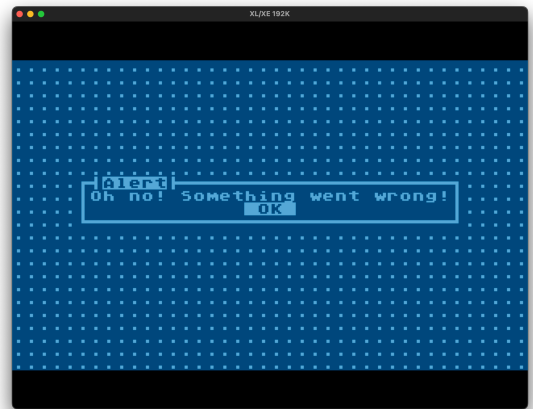
A8LIBMISC.C - WaitKCX()

### 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.



---

byte GButton(byte bN, byte x, byte y, byte bD, byte bS  
unsigned char \*\*pA)

Parameters: bN = Window handle number  
x = Window column to start get  
y = Window row  
bD = Initial selected button  
GDISP = Display only and exit  
bS = Number of buttons in array pA  
pA = Pointer to ragged array of button name strings

Returns: byte = Button number selected or exit status  
XESC = Exited with ESCape (no button chosen)  
XTAB = Exited with TAB (no button chosen)

Requires: A8DEFINES.H  
A8DEFWIN.H  
A8LIBWIN.C - WPrint()  
A8LIBMISC.C - WaitKCX()  
Standard - strlen()

### Description

Displays a row of buttons and gets selection from user.

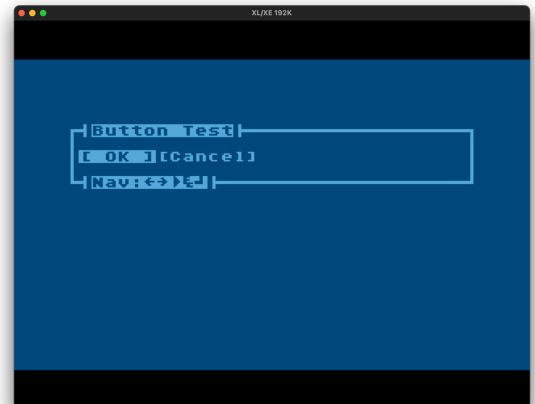
If the initial selection indicator (bD) is passed as GDISP, then the buttons will be displayed and the function will exit (none will be highlighted).

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

Parameters: bN = Window handle number  
x = Window column to start get  
y = Window row  
bI = Display Only indicator  
    GDISP = Display only and exit  
    GEDIT = Edit checkbox  
bD = Default initial value  
    GCON = Checked  
    GCOFF = Unchecked

Returns: byte = Checked status or exit status  
    GCON = Checked  
    GCOFF = Not Checked  
    XESC = Exited with ESCape (no value chosen)  
    XTAB = Exited with TAB (no value chosen)

Requires: A8DEFINES.H  
    A8DEFWIN.H  
    A8LIBWIN.C - WPrint()  
    A8LIBMISC.C - WaitKCX()

### Description

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

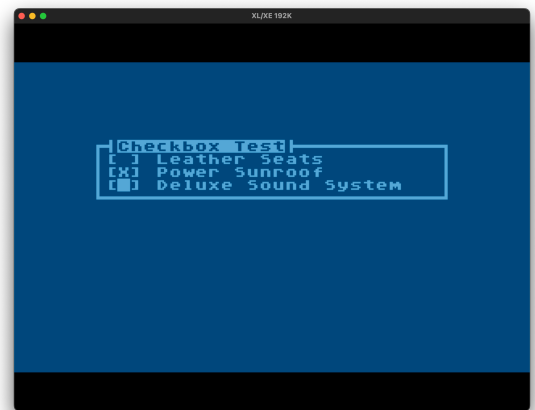
Unlike many other input gadgets, the text for the option is not included and should be displayed separately in the window using WPrint() prior to calling GCheck().

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.

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 GInput(byte bN, byte x, byte y, byte bT, byte bS,  
unsigned char *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 (max 128)

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

Requires: **A8DEFINES.H**  
**A8DEFWIN.H**  
**A8LIBMISC.C**     - `WaitKCX()`, `IKC2ATA()`  
**A8LIBWIN.C**     - `WPrint()`  
Standard         - `memset()`, `strlen()`, `strcpy()`, `strncpy()`

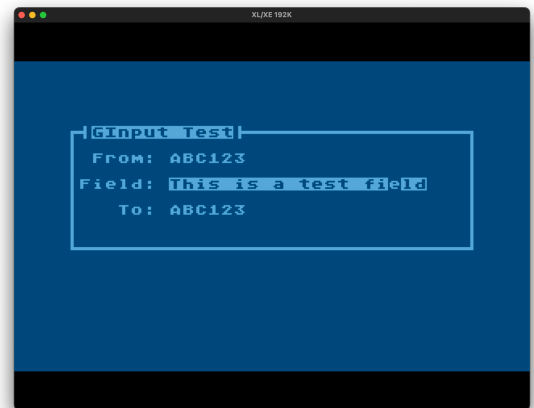
### Description

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`.

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 unsigned 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 GProg(byte bN, byte x, byte y, byte bS)`

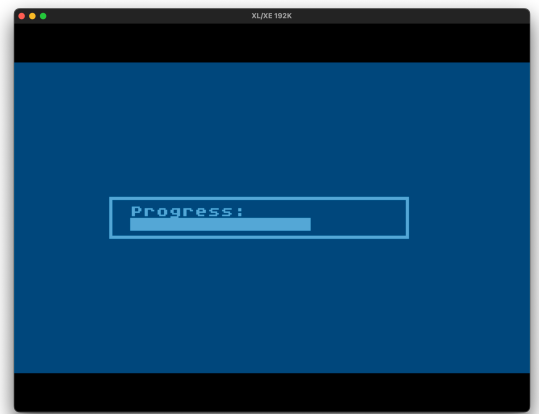
Parameters: `bN` = Window handle number  
`x` = Window column to display bar at  
`y` = Window row to display bar at  
`bS` = Bar size (Percent complete), Range 0 to 100.

Returns: `void`

Requires: `A8DEFINES.H`  
`A8DEFWIN.H`  
`A8LIBWIN.C` - `WPrint()`  
Standard - `memset()`, `strcpy()`

### 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`, and has a range of 0 to 100. The bar will always be displayed with 20 characters of length, so the window it is placed in should be a minimum of 22 wide to account for the frames.



---

`byte GRadio(byte bN, byte x, byte y, byte bD, byte bE, byte bI, byte bS, unsigned char **pS)`

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)  
`bE` = Display Only indicator  
    GDISP = Display only and exit  
    GEDIT = Edit  
`bI` = Initial selected button  
`bS` = Number of buttons in array `pS`  
`pS` = Pointer to ragged array of button name strings

Returns: `byte` = Button number selected or exit status  
    XESC = Exited with ESCape (no button chosen)  
    XTAB = Exited with TAB (no button chosen)

Requires: `A8DEFINES.H`  
`A8DEFWIN.H`  
`A8LIBWIN.C` - `WPrint()`, `WPos()`, `WPut()`  
`A8LIBMISC.C` - `WaitKCX()`  
Standard - `strlen()`

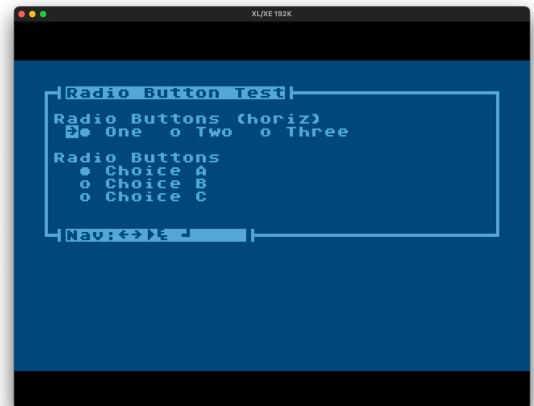
### Description

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

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 GSpin(byte bN, byte x, byte y, byte bL, byte bM, byte bI, byte bE)

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  
bI = Initial value  
bE = Display only indicator  
GDISP = Display only and exit  
GEDIT = Edit

Returns: byte = Value selected or exit status  
XESC = Exited with ESCape (no value chosen)  
XTAB = Exited with TAB (no value chosen)

Requires: A8DEFINES.H  
A8DEFWIN.H  
A8LIBWIN.C - WPrint()  
A8LIBMISC.C - WaitKCX()  
Standard - sprintf()

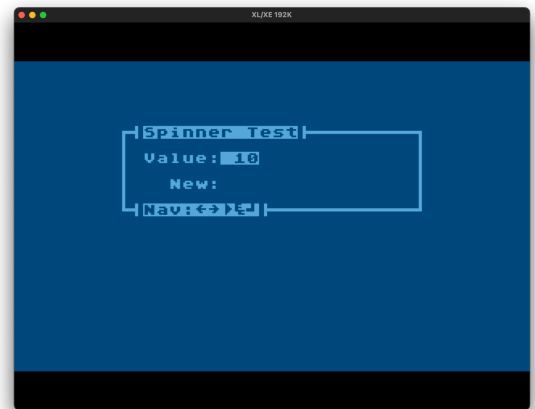
### Description

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 250, and is limited by the function at this value. 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 it is a hybrid. It will return those values, and it returns the selected value. Realistically, the foreseen use case is from 0 to 100.

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 (A8LIBMENU.C)

byte MenuV(byte bN, byte x, byte y, byte bI, byte bS, byte bC,  
unsigned char \*\*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  
bS = Start item selection number  
bC = Number of menu items in array pS  
pS = Pointer to array of strings containing menu items

Returns: byte = Number of item chosen or exit value  
XESC = Exited with ESCape (no item chosen)  
XTAB = Exited with TAB (no item chosen)

Requires: A8DEFINES.H  
A8DEFWIN.H  
A8LIBWIN.C - WPrint()  
A8LIBMISC.C - WaitKCX()  
Standard - strcpy()

### Description

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

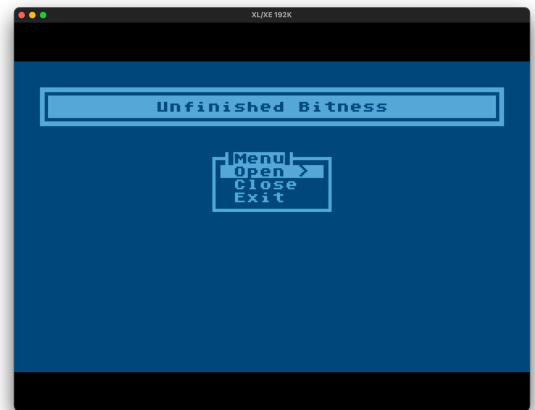
The currently selected menu item will be highlighted (displayed in inverse video), while the remaining items will be in normal video.

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*).



## String Manipulation (A8LIBSTR.C)

---

`void StrAI(unsigned char *pS, byte bS)`

Parameters: pS = Pointer to text string  
bS = Size (number) of chars in string to convert  
Returns: void  
Requires: n/a

### Description

Converts elements (characters) of the string referenced by pS from the **ATASCII** code representation to the internal code representation up to size bS bytes in length.

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

This is the opposite of StrIA().

---

`void StrIA(unsigned char *pS, byte bS)`

Parameters: pS = Pointer to text string  
bS = Size (number) of chars in string to convert  
Returns: void  
Requires: n/a

### Description

Converts elements (characters) of the string referenced by pS from the internal code representation to the **ATASCII** code representation up to size bS bytes in length.

This is the opposite of StrAI().

---

`void StrInv(unsigned char *pS, byte bS)`

Parameters: pS = Pointer to text string  
bS = Size (number) of chars in string to inverse  
Returns: void  
Requires: n/a

### Description

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

## APE Atari Peripheral Emulator (A8LIBAPE.C)

---

`void ApeTimeG(char *cB)`

Parameters: cB = Pointer to text string to hold date and time

Returns: void

Requires: n/a

### **Description**

Queries the SIO chain for the APE device and executes the APE get time call. The result is 6 bytes referencing DMYHMS which is stored in the string referenced by cB.

The century for the year (Y) is expected to be 2000, and the value will be that of the remaining 2 digit year. Example: 2023 will have the value of 23.

When using this function, make sure you do not have multiple devices in the SIO chain that support the APE Time protocol. Strange results will occur!

## FujiNet (A8LIBFUJI.C)

---

### unsigned short FNCheck(void)

Parameters: N/A  
Returns: unsigned short = Number of bytes waiting  
Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H

#### **Description**

Queries the FujiNet connection for incoming bytes waiting to be read, and returns the result.

---

### unsigned char FNClose(void)

Parameters: N/A  
Returns: unsigned char = SIO Status  
FNSOK = Success  
Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H

#### **Description**

Closes the current connection.

---

## unsigned char FNGConfig(td\_fncfg \*sC)

Parameters: sC = Pointer to memory location of a valid FujiNet configuration structure data type

Returns: unsigned char = SIO Status  
FNSOK = Success

Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H

### Description

Reads the configuration data of the FujiNet device and places it in the data structure pointed to sC. A variable for the structure can be instantiated with the td\_fncfg data type, and should be passed to FNGConfig by address.

There are 140 bytes in the structure which is comprised of the following fields, all of which are character arrays:

ssid	: 33 bytes	= Connected network SSID name
hostname	: 64 bytes	= Hostname of the FujiNet device
ip	: 4 bytes	= IP address of the FujiNet device
gateway	: 4 bytes	= Gateway address
netmask	: 4 bytes	= Network mask
dns	: 4 bytes	= DNS resolver IP address
mac	: 6 bytes	= MAC address of the FujiNet device
bssid	: 6 bytes	= MAC address of the connected Wi-Fi router
version	: 15 bytes	= FujiNet firmware version number

---

## unsigned char FNOpen(char \*cB)

Parameters: cB = Pointer to string containing connection spec

Returns: unsigned char = SIO Status  
FNSOK = Success

Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H

### Description

Opens a connection to the address specified in the string pointer cB. cB should point to a character string that *MUST* be 256 bytes in length.

The format of the connection string is:

N:<protocol>:<address>[:<port>][</path>]

N:	=	Device Name ( <i>required</i> )
protocol	=	Typical web protocol selection ( <i>required</i> )
	HTTP://	= HyperText Transfer Protocol
	HTTPS://	= HyperText Transfer Protocol w/SSL
	TCP://	= Transmission Control Protocol (raw)
	UDP://	= User Datagram Protocol (raw)
	FTP://	= File Transfer Protocol
	TNFS://	= Tiny Network File System
address	=	IP address or resolvable host name ( <i>required</i> )
:port	=	Port to connect on (optional)
/path	=	Path to resource (optional)

---

## unsigned char FNRead(unsigned short iS, unsigned char \*cB)

Parameters: iS = Number of bytes to read  
cB = Pointer to memory location to place read data

Returns: unsigned char = SIO Status  
FNSOK = Success

Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H

### Description

Reads iS bytes from the currently open connection and places the data in the pointer referenced by cB.

---

## unsigned char FNStatus(void)

Parameters: N/A

Returns: unsigned char = Wi-Fi Status  
FNDSIDLE = Wi-Fi is idle  
FNDSNOSSID = No SSID available  
FNDSSCANC = Scan completed  
FNDSACTIVE = Active, connected to Wi-Fi network  
FNDSLFAIL = Last connect failed  
FNDSWIFILOST = Wi-Fi connection lost  
FNDSWIFIDISC = Wi-Fi explicitly disconnected

Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H

### Description

Returns the FujiNet's Wi-Fi status.

---

## void FNTrans(unsigned char bM, unsigned char bD)

Parameters: bM = Translation mode desired  
FNTRNONE = No translation  
FNTRCR = Carriage Return to EOL  
FNTRLF = Line Feed to EOL  
FNTRCRLF = Carriage Return & Line Feed to EOL  
bD = Direction of translation  
FNAXREAD = Reading from connection  
FNAXWRITE = Writing to connection  
FNAXUPDATE = Bi-directional

Returns: N/A

Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H

### Description

Returns the FujiNet's **ASCII** EOL translation mode.

---

## `unsigned char FNWrite(unsigned char *cB)`

Parameters: cB = Pointer to memory location of data to write

Returns: unsigned char = SIO Status  
FNSOK = Success

Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H

### **Description**

Writes the data from the pointer referenced by cB to the currently open connection.

## FujiNet Base64 (A8LIBFUJIB64.C)

The Base64 commands have a particular order in which they should be called. See the demonstration application for a complete example. In short, that order, regardless of the operation being an encode or decode, is:

FNB64Inp() = Set the input buffer

FNB64Cmp() = Compute (Perform) the encode/decode

FNB64Len() = Get the length of the result

FNB64Out() = Get the output buffer

---

### unsigned char FNB64Cmp(char cT)

Parameters: cT = Operation type (encode or decode)

FNLENCODE = Encode

FNLDECODE = Decode

Returns: unsigned char = SIO Status

FNSOK = Success

Requires: A8DEFINES.H

A8DEFSIO.H

A8DEFFUJI.H

A8DEFFUJIB64.H

#### Description

Performs the computation of the encode or decode on the previously set input buffer. Decoding or Encoding is selected by means of cT.



---

## unsigned char FNB64Inp(char cT, char \*cB, unsigned short iS)

Parameters: cT = Operation type (encode or decode)  
              FNLENCODE = Encode  
              FNLDECODE = Decode  
              cB = Pointer to text to load into encode or decode input buffer  
              iS = Number of bytes to load into encode or decode input buffer (size of cB)

Returns: unsigned char = SIO Status  
          FNSOK = Success

Requires: A8DEFINES.H  
          A8DEFSIO.H  
          A8DEFFUJI.H  
          A8DEFFUJIB64.H

### Description

Loads the FujiNet's Base64 encode or decode input buffer with the string pointed to by cB. The decode or encode input buffer is selected by means of cT. The length of the text to load into the encode or decode input buffer is specified via iS.

---

## unsigned char FNB64Len(char cT, unsigned long \*iS)

Parameters: cT = Operation type (encode or decode)  
              FNLENCODE = Encode  
              FNLDECODE = Decode

Returns: unsigned char = SIO Status  
          FNSOK = Success

Requires: A8DEFINES.H  
          A8DEFSIO.H  
          A8DEFFUJI.H  
          A8DEFFUJIB64.H

### Description

Gets the length of the result of the encode or decode of the previously set performed compute. The decode or encode output buffer is selected by means of cT. The buffer length is stored in the long pointed to by iS. The maximum value is 65,535.

---

`unsigned char FNB64Out(char cT, char *cB, unsigned short iS)`

Parameters: `cT` = Operation type (encode or decode)  
              `FNLENCODE` = Encode  
              `FNLDECODE` = Decode  
              `cB` = Pointer to text to encode or decode  
              `iS` = Number of bytes to encode or decode (size of `cB`)

Returns: `unsigned char` = SIO Status  
          `FNSOK` = Success

Requires: `A8DEFINES.H`  
          `A8DEFSIO.H`  
          `A8DEFFUJI.H`  
          `A8DEFFUJIB64.H`

### **Description**

Read the result from the last encode or decode operation from the FujiNet's Base64 encode or decode output buffer into the string pointed to by `cB`. The decode or encode output buffer is selected by means of `cT`. The length of the text to retrieve from the encode or decode output buffer is specified via `iS`.

## FujiNet Hash (A8LIBFUJIHASH.C)

The Hash commands have a particular order in which they should be called. See the demonstration application for a complete example. In short, that order is:

FNHashInp()       = Set the input buffer  
FNHashCmp()       = Compute (Perform) the hash  
FNHashOut()       = Get the output buffer

Note there is no length function. The SHA hashes have a preset output length.

---

### unsigned char FNHashCmp(byte bT)

Parameters: bT = Hash type

FNHASHMD5   = MD5 (currently not implement within FujiNet firmware)  
FNHASHSHA1   = SHA1  
FNHASHSHA2   = SHA256  
FNHASHSHA5   = SHA512

Returns: unsigned char = SIO Status  
FNSOK = Success

Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H  
A8DEFFUJIHASH.H

#### Description

Performs the hash computation on the previously set hash input buffer. The hash type is selected by means of bT.

---

## unsigned char FNHashInp(char \*cB, unsigned short iS)

Parameters: cB = Pointer to text to load into hash input buffer  
iS = Number of bytes to load into hash input buffer (size of cB)

Returns: unsigned char = SIO Status  
FNSOK = Success

Requires: A8DEFINES.H  
A8DEFSIO.H  
A8DEFFUJI.H  
A8DEFFUJIB64.H

### **Description**

Loads the FujiNet's hash input buffer with the string pointed to by cB. The length of the text to load into the hash input buffer is specified via iS.

---

## unsigned char FNHashOut(byte bT, byte bH, char \*cB)

Parameters: bT = Hash type

FNHASHMD5 = MD5 (currently not implement within FujiNet firmware)

FNHASHSHA1 = SHA1

FNHASHSHA2 = SHA256

FNHASHSHA5 = SHA512

bH = Output type

FNHASHOBIN = Binary

FNHASHOHEX = Hexadecimal

cB = Pointer to character buffer to hold hash

Returns: unsigned char = SIO Status

FNSOK = Success

Requires: A8DEFINES.H

A8DEFSIO.H

A8DEFFUJI.H

A8DEFFUJIB64.H

### Description

Read the result from the last hash operation from the FujiNet's hash output buffer into the string pointed to by cB. The hash type is selected by means of bT. The output type is selected with bH, and can be binary or hexadecimal. Both the hash type and output need to be specified so the proper size to retrieve can be computed.

The output buffer should be sized large enough to hold the hash. The size varies by hash type and output type. Use this table to determine size:

Hash Type	Output Binary Size	Output Hexadecimal Size
FNHASHMD5	16	32
FNHASHSHA1	20	40
FNHASHSHA2	32	64
FNHASHSHA5	64	128

## Miscellaneous (A8LIBMISC.ACT)

---

### byte 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: A8DEFINES.H

#### 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.

---

### byte WaitYN(byte bD)

Parameters: bD = Flag for display of ? prompt  
WON = Display ? prompt  
WOFF = Do not display ? prompt

Returns: byte = 1 for Y or y pressed  
0 for N or n pressed

Requires: A8DEFINES.H  
Standard - PEEK(), POKE(), cputc()

#### 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 bD is set to WON.

The keypress is consumed before returning.

---

## word WaitKCX(byte bI)

Parameters: bI = Flag to execute inverse function or not

WON = Yes

WOFF = No

Returns: word = key code value of key pressed

KFHLP = Help key

KF1 = F1 key (1200XL/1400XL/1450XLD only)

KF2 = F2 key (1200XL/1400XL/1450XLD only)

KF3 = F3 key (1200XL/1400XL/1450XLD only)

KF4 = F4 key (1200XL/1400XL/1450XLD only)

KCAP = Caps key

KINV = Inverse key

KCOPT = Option key

KCSEL = Select key

KCSTA = Start key

Requires: A8DEFINES.H

Standard - PEEK(), POKE()

### 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 WON.

The keypress is consumed or debounced as necessary before the function returns.

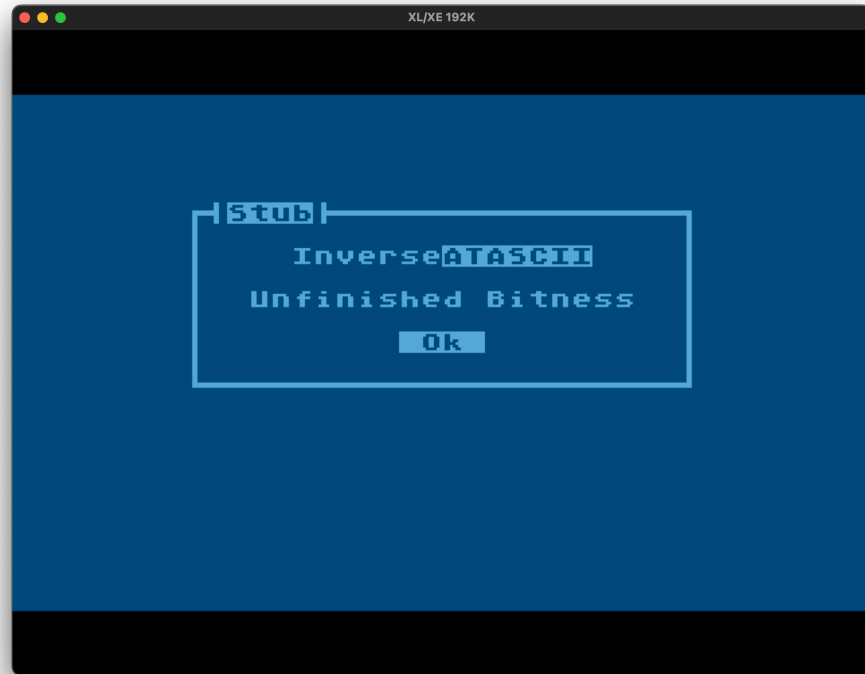
This is intended for use on XL/XE computers.

# 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.



File: STUBWIN.C

```
// -----  
// Program: stubwin.c  
// Desc...: A8 Library Stub Window Program  
// Author.: Wade Ripkowski  
// Date...: 20220826  
// Notes..: cl65 -v [-O] -t atarixl stubwin.c -o stubwin.xex  
// -----  
  
// Pull in include files  
#include <stdio.h>  
#include <conio.h>  
#include <string.h>  
#include <atari.h>  
  
#include "a8defines.h"  
#include "a8defwin.h"
```



```

#include "a8libstr.c"
#include "a8libwin.c"
#include "a8libmisc.c"

// -----
// Func...: void main(void)
// Desc...: Main routine
// -----
void main(void)
{
    // Variables
    byte bW1;

    // Init Window System
    WInit();

    // Open window
    bW1 = WOpen(8, 5, 24, 9, WOFF);
    WOrn(bW1, WPTOP, WPLFT, "Stub");
    WPrint(bW1, 5, 2, WOFF, "Inverse");
    WPrint(bW1, 12, 2, WON, "ATASCII");
    WPrint(bW1, WPCNT, 4, WOFF, "Unfinished Bitness");
    WPrint(bW1, WPCNT, 6, WON, " Ok ");

    // Wait for a keystroke or console key
    WaitKCX(WOFF);

    // Close window
    WClose(bW1);

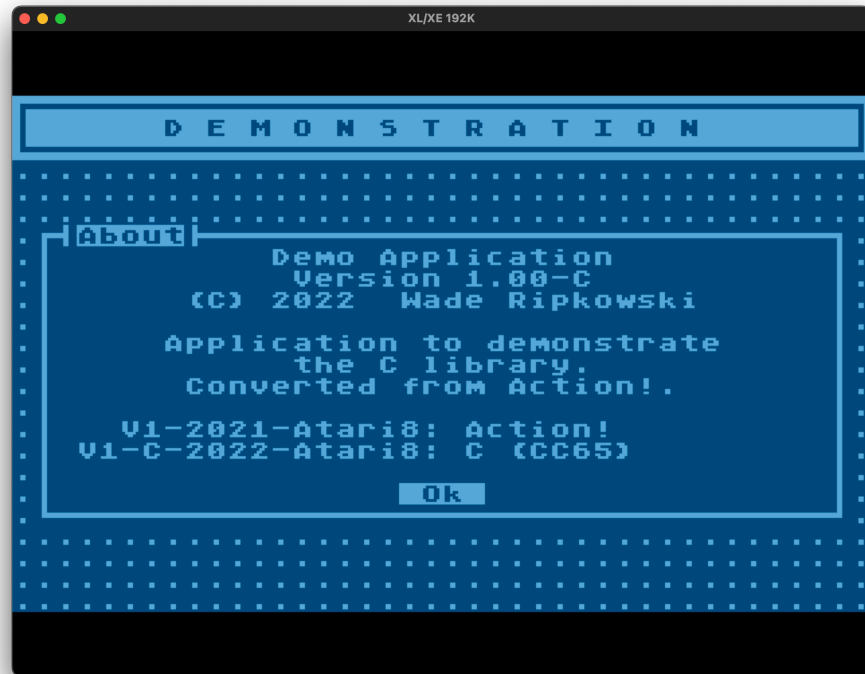
    // Exit
    return;
}

```

---

## 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.



File: STUBAPP.C

```
// -----  
// Program: stubapp.c  
// Desc...: A8 Library Stub Application  
// Author.: Wade Ripkowski  
// Date...: 20220826  
// Notes..: cl65 -v [-O] -t atarixl stubapp.c -o stubapp.xex  
// -----  
  
// Pull in include files  
#include <stdio.h>  
#include <conio.h>  
#include <unistd.h>  
#include <string.h>  
#include <atari.h>  
  
#include "a8defines.h"  
#include "a8defwin.h"  
#include "a8libmisc.c"  
#include "a8libstr.c"  
#include "a8libwin.c"  
#include "a8libgadg.c"  
#include "a8libmenu.c"
```

```

// Prototypes
void About(void);
void SubMenu3(void);

// -----
// Func...: void About(void)
// Desc...: About Dialog
// -----
void About(void)
{
    byte bW1;

    // Show window
    bW1 = WOpen(1, 6, 38, 14, WOFF);
    WOrn(bW1, WPTOP, WPLFT, "About");
    WPrint(bW1, WPCNT, 1, WOFF, "Demo Application");
    WPrint(bW1, WPCNT, 2, WOFF, "Version 1.00-C");
    WPrint(bW1, WPCNT, 3, WOFF, "(C) 2022 Wade Ripkowski");
    WPrint(bW1, WPCNT, 5, WOFF, "Application to demonstrate");
    WPrint(bW1, WPCNT, 6, WOFF, "the C library.");
    WPrint(bW1, WPCNT, 7, WOFF, "Converted from Action!.");
    WPrint(bW1, 4, 9, WOFF, "V1-2021-Atari8: Action!");
    WPrint(bW1, 2, 10, WOFF, "V1-C-2022-Atari8: C (CC65)");
    WPrint(bW1, WPCNT, 12, WON, " Ok ");

    // Wait for key
    WaitKCX(WOFF);

    // Close window
    WClose(bW1);

    return;
}

// -----
// Func...: void SubMenu3(void)
// Desc...: Sub menu routine
// -----
void SubMenu3(void)
{
    byte bW1, bC;
    byte bD = FALSE;
    unsigned char *pcM[4] = { " Sub-Item 1 ", " Sub-Item 2 ", " Sub-Item 3 " };

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

    // Loop until exit
    while (! bD)
    {
        // Display menu and get choice
        bC = MenuV(bW1, 1, 1, WOFF, 1, 3, pcM);
    }
}

```

```

// Process choice
switch (bC)
{
    case 1: GAlert(" Sub-Item 1 selected. ");
            break;

    case 2: GAlert(" Sub-Item 2 selected. ");
            break;

    case 3: GAlert(" Sub-Item 3 selected. ");
            break;

    case XESC: bD = TRUE;
              break;
}
}

// Close window
WClose(bW1);

return;
}

// -----
// Func...: void main(void)
// Desc...: Main routine
// -----
void main(void)
{
    // Variables
    byte bW1, bW2, bC;
    byte bD = FALSE;
    unsigned char *pcM[6] =
        { " Sub-Menu 1 ", " Sub-Menu 2 ", " Sub-Menu 3 ", " About      ", " Exit      ", " };

    // Setup screen
    WInit();

    // Set Background
    WBack(14);

    // Open header window
    bW1 = WOpen(0, 0, 40, 3, WON);
    WPrint(bW1, WPCNT, 1, WOFF, "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");

    // Loop until done (Exit selected)
    while (! bD) {
        // Call menu
        bC = MenuV(bW2, 1, 2, WOFF, 1, 5, pcM);
    }
}

```

```

// Process choice
switch (bC)
{
    case 1: GAlert(" Sub-Menu 1 selected. ");
            break;

    case 2: GAlert(" Sub-Menu 2 selected. ");
            break;

    case 3: SubMenu3();
            break;

    case 4: About();
            break;

    case 5: bD = TRUE;
            break;
}

// Exit on ESC as well
if (bC == XESC)
{
    bD = TRUE;
}
}

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

// Exit
return;
}

```

## Demo Programs

### 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.



File: APPDEMO.C

```
// -----  
// Program: appdemo.c  
// Desc...: A8 Library Demo Application  
// Author.: Wade Ripkowski  
// Date...: 20220825  
// Notes..: cl65 -v [-O] -t atari appdemo.c -o appdemo.xex  
// -----  
  
// Pull in include files  
#include <stdio.h>  
#include <conio.h>  
#include <unistd.h>  
#include <string.h>  
#include <atari.h>  
  
#include "a8defines.h"  
#include "a8defwin.h"  
#include "a8libmisc.c"  
#include "a8libstr.c"
```

```

#include "a8libwin.c"
#include "a8libgadg.c"
#include "a8libmenu.c"

// Prototypes
void DoSpin(void);
byte FormInput(void);
void ProgTest(void);
void About(void);

// -----
// Func...: void FormInput(void)
// Desc...: Demo use of input gadgets
// Returns: TRUE if accepted, else FALSE
// Notes...: Maximum local variable stack space is 256 bytes.
//           MUST use pragma static-locals to move variables to
//           BSS segment due to total size in this function.
// -----
#pragma static-locals(push, on)
byte FormInput(void)
{
    byte bR = FALSE, bRA = 1, bRB = 1, bChap = GCOFF, bChbp = GCON, bChcp = GCOFF, bV = 10;
    byte bW1, bM, bA, bB, bC, bD, bVp, bRAp, bRBp, bCha, bChb, bChc;
    // Regular buttons, radio buttons, and data field names
    unsigned char *paB[3] = { "[ Ok ]", "[Cancel]" },
        *prA[4] = { "One", "Two", "Three" },
        *prB[4] = { "Choice A", "Choice B", "Choice C" },
        *paD[5] = { "Numer", "Alpha", "AlNum", "Any.." };
    // Input strings & navigation strings
    unsigned char cA[41], cB[41], cC[41], cD[41],
        cF[15], cI[15], cR[15], cX[15];

    // Define navigation strings
    sprintf(cF, "Nav:%c%c%c%c%c%c ", CHUP, CHDN, CHLFT, CHRGT, CHTAB, CHESC, CHBTRGT);
    sprintf(cI, "Nav:%c%c%c%c^cS^cE", CHLFT, CHRGT, CHESC, CHBTRGT);
    sprintf(cR, "Nav:%c%c%c%c%c%c %c ", CHUP, CHDN, CHLFT, CHRGT, CHTAB, CHESC, CHBTRGT);
    sprintf(cX, "Nav:X %c%c%c ", CHTAB, CHESC, CHBTRGT);

    // Define input string defaults
    strcpy(cA, "-100.00");
    strcpy(cB, "This string has something to edit in it!");
    strcpy(cC, "");
    sprintf(cD, "%cAny character string!%c", CHBALL, CHBALL);

    // Set radio button and spinner previous selection defaults
    bRAp = bRA;
    bRBp = bRB;
    bVp = bV;

    // Open window & draw form
    bW1 = WOpen(2, 4, 36, 18, WOFF);
    WOrn(bW1, WPTOP, WPLFT, "Input Form");
    WOrn(bW1, WPTOP, WPRGT, "Edit");
    WOrn(bW1, WPBOT, WPLFT, cF);

```

```

WPrint(bW1, 1, 1, WOFF, "Data Fields");
WPrint(bW1, 2, 2, WOFF, "Numer:");
WPrint(bW1, 2, 3, WOFF, "Alpha:");
WPrint(bW1, 2, 4, WOFF, "AlNum:");
WPrint(bW1, 2, 5, WOFF, "Any..:");
WPrint(bW1, 2, 6, WOFF, "Spin.:");
GSpin(bW1, 8, 6, 0, 100, bVp, GDISP);

WPrint(bW1, 1, 8, WOFF, "Radio Buttons (h)");
GRadio(bW1, 2, 9, GHORZ, GDISP, bRAp, 3, prA);

WPrint(bW1, 1, 11, WOFF, "Radio Buttons (v)");
GRadio(bW1, 2, 12, GVERT, GDISP, bRBp, 3, prB);

WPrint(bW1, 20, 11, WOFF, "Check Boxes");
WPrint(bW1, 25, 12, WOFF, "Milk");
WPrint(bW1, 25, 13, WOFF, "Bread");
WPrint(bW1, 25, 14, WOFF, "Butter");
GCheck(bW1, 21, 12, GDISP, bChap);
GCheck(bW1, 21, 13, GDISP, bChbp);
GCheck(bW1, 21, 14, GDISP, bChcp);

GButton(bW1, 21, 16, GDISP, 2, paB);

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

// Loop until form accepted
do
{
    // ----- Display Input Fields -----
    // Show navigation info
    WOrn(bW1, WPBOT, WPLFT, cI);

    // Edit fields
    bA = GInput(bW1, 8, 2, GNUMER, 27, cA);
    bB = GInput(bW1, 8, 3, GALPHA, 27, cB);
    bC = GInput(bW1, 8, 4, GALNUM, 27, cC);
    bD = GInput(bW1, 8, 5, GANY, 27, cD);

    // ----- Spinner Input -----
    bV = GSpin(bW1, 8, 6, 0, 100, bVp, GEDIT);
    if ((bV != XESC) && (bV != XTAB))
    {
        bVp = bV;
    }

    // ----- Display Radio Buttons - horizontal -----
    // Show navigation info
    WOrn(bW1, WPBOT, WPLFT, cR);

    // Process buttons
    bRA = GRadio(bW1, 2, 9, GHORZ, GEDIT, bRAp, 3, prA);

```



```

// If not bypass, set previous selected value
if ((bRA != XESC) && (bRA != XTAB))
{
    bRAp = bRA;
}

// Redisplay buttons
GRadio(bW1, 2, 9, GHORZ, GDISP, bRAp, 3, prA);

// ----- Display Radio Buttons - vertical -----
bRB = GRadio(bW1, 2, 12, GVERT, GEDIT, bRBp, 3, prB);

// If not bypass, set previous selected value
if ((bRB != XESC) && (bRB != XTAB))
{
    bRBp = bRB;
}

// Redisplay buttons
GRadio(bW1, 2, 12, GVERT, GDISP, bRBp, 3, prB);

// ----- Display 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, 12, GEDIT, bChap);

    // If not ESC or TAB, set previous value
    if ((bCha != XESC) && (bCha != XTAB))
    {
        bChap = bCha;
    }
} while ((bCha != XESC) && (bCha != XTAB));

// Stay on this check until ESC, TAB, or set
do
{
    // Display button and get choice
    bChb = GCheck(bW1, 21, 13, GEDIT, bChbp);

    // If not ESC or TAB, set previous value
    if ((bChb != XESC) && (bChb != XTAB))
    {
        bChbp = bChb;
    }
} while ((bChb != XESC) && (bChb != XTAB));

// Stay on this check until ESC, TAB, or set
do
{
    // Display button and get choice

```

```

        bChc = GCheck(bW1, 21, 14, GEDIT, bChcp);

        // If not ESC or TAB, set previous value
        if ((bChc != XESC) && (bChc != XTAB))
        {
            bChcp = bChc;
        }
    } while ((bChc != XESC) && (bChc != XTAB));

    // Set footer
    WOrn(bW1, WPBOT, WPLFT, cF);

    // Prompt to accept form and redisplay buttons
    bM = GButton(bW1, 21, 16, GEDIT, 2, paB);
    GButton(bW1, 21, 16, GDISP, 2, paB);
} while (bM == XTAB);

// Check for acceptance (OK button), and set exit flag
if (bM == 1)
{
    bR = TRUE;
    GAlert("Doing something with entered data...");
}

// Close window
WClose(bW1);

return(bR);
}
#pragma static-locals(pop)

// -----
// Func...: void ProgTest(void)
// Desc...: Demos window status and progress bar.
// -----
void ProgTest(void)
{
    byte bW1, bW2, bL, bS;
    unsigned int iV;

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

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

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

    // Loop through each window handle
    for (bL=0; bL < 10; bL+=1)

```

```

{
    // Get the window status
    bS = WStat(bL);

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

    // Print the window handle status
    WPrint(bW1, 8, 3+bL, WOFF, (bS == WON ? "Used" : "Free"));

    // Update progress bar
    iV = ((bL + 1) % 10) * 10;
    if (iV == 0)
    {
        iV = 100;
    }
    GProg(bW2, 2, 2, (byte) iV);

    // Wait 1 second
    sleep(1);
}

GAlert(" Press a key to continue. ");

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

return;
}

// -----
// Func...: void About(void)
// Desc...: About Dialog
// -----
void About(void)
{
    byte bW1;

    // Show window
    bW1 = WOpen(1, 6, 38, 14, WOFF);
    WOrn(bW1, WPTOP, WPLFT, "About");
    WPrint(bW1, WPCNT, 1, WOFF, "Demo Application");
    WPrint(bW1, WPCNT, 2, WOFF, "Version 1.00-C");
    WPrint(bW1, WPCNT, 3, WOFF, "(C) 2022 Wade Ripkowski");
    WPrint(bW1, WPCNT, 5, WOFF, "Application to demonstrate");
    WPrint(bW1, WPCNT, 6, WOFF, "the C library.");
    WPrint(bW1, WPCNT, 7, WOFF, "Converted from Action!.");
    WPrint(bW1, 4, 9, WOFF, "V1-2021-Atari8: Action!");
    WPrint(bW1, 2, 10, WOFF, "V1-C-2022-Atari8: C (CC65)");
    WPrint(bW1, WPCNT, 12, WON, " Ok ");

    // Wait for key
    WaitKCX(WOFF);

```

```

    // Close window
    WClose(bW1);

    return;
}

// -----
// Func...: void SubMenu(void)
// Desc...: Sub menu routine
// -----
void SubMenu(void)
{
    byte bW1, bC;
    byte bD = FALSE;
    unsigned char *pcM[4] = { " Sub-Item 1 ", " Sub-Item 2 ", " Sub-Item 3 " };

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

    // Loop until exit
    while (! bD)
    {
        // Display menu and get choice
        bC = MenuV(bW1, 1, 1, WOFF, 1, 3, pcM);

        // Process choice
        switch (bC)
        {
            case 1: GAlert(" Sub-Item 1 selected. ");
                    break;

            case 2: GAlert(" Sub-Item 2 selected. ");
                    break;

            case 3: GAlert(" Sub-Item 3 selected. ");
                    break;

            case XESC: bD = TRUE;
                      break;
        }
    }

    // Close window
    WClose(bW1);

    return;
}

// -----
// Func...: void main(void)
// Desc...: Main routine
// -----

```

```

void main(void)
{
    // Variables
    byte bW1, bW2, bC;
    byte bD = FALSE;
    unsigned char *pcM[6] =
        { " Input Form ", " Progress Bar ", " Sub-Menu ", " About ", " Exit " };

    // Setup screen
    WInit();
    WBack(14);

    // Open header window
    bW1 = WOpen(0, 0, 40, 3, WON);
    WPrint(bW1, WPCNT, 1, WOFF, "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, WPCNT, "Menu");

    // Loop until done (Exit selected)
    while (! bD) {
        // Call menu
        bC = MenuV(bW2, 1, 2, WON, 1, 5, pcM);

        // Process choice
        switch (bC)
        {
            case 1: FormInput();
                    break;

            case 2: ProgTest();
                    break;

            case 3: SubMenu();
                    break;

            case 4: About();
                    break;

            case 5: bD = TRUE;
                    break;
        }

        // Exit on ESC as well
        if (bC == XESC)
        {
            bD = TRUE;
        }
    }

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

    // Exit

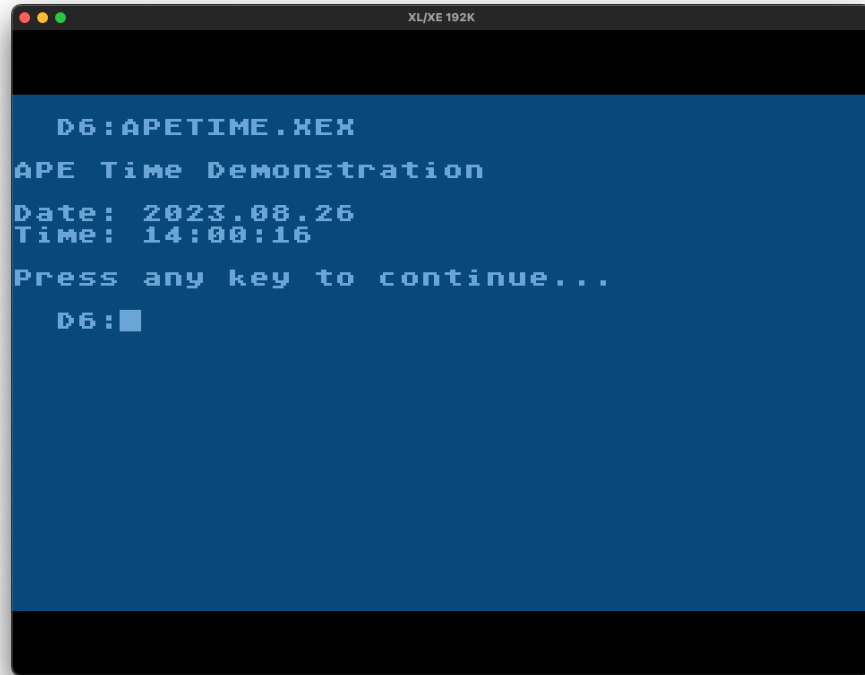
```

```
    return;  
}
```

---

## Demonstration APE Time Application

This is a very simple demonstration of a call to the APE device to get the current date and time, then displays it on the screen. It shows how to include the library. It does not use the TUI (text user interface) components or the FujiNet components of the library.



File: APETIME.C

```
// -----  
// Program: apetime.c  
// Desc...: Gets date/time from APE (via FujiNet or other APE device)  
// Author.: Wade Ripkowski  
// Date...: 2023.02.09  
// Notes...: cl65 -v [-O] -t atari apetime.c -o apetime.xex  
// -----  
  
// Pull in include files  
#include <stdio.h>  
#include <atari.h>  
#include <conio.h>  
#include <unistd.h>  
  
#include "a8defsio.h"  
#include "a8libape.c"  
  
// -----  
// Main Routine  
// -----
```

```

void main(void)
{
    // Storage for 6 bytes preset to 0
    char bDT[6]={ 0, 0, 0, 0, 0, 0 };

    // Call time routine with the storage array
    ApeTimeG(bDT);

    // What time is it?
    printf("\nAPE Time Demonstration\n\n");
    printf("Date: 20%d.%02d.%02d\n", bDT[2], bDT[1], bDT[0]);
    printf("Time: %02d:%02d:%02d\n\n", bDT[3], bDT[4], bDT[5]);

    // Wait for a keystroke
    printf("Press any key to continue...\n");
    while (! kbhit()) {};
}

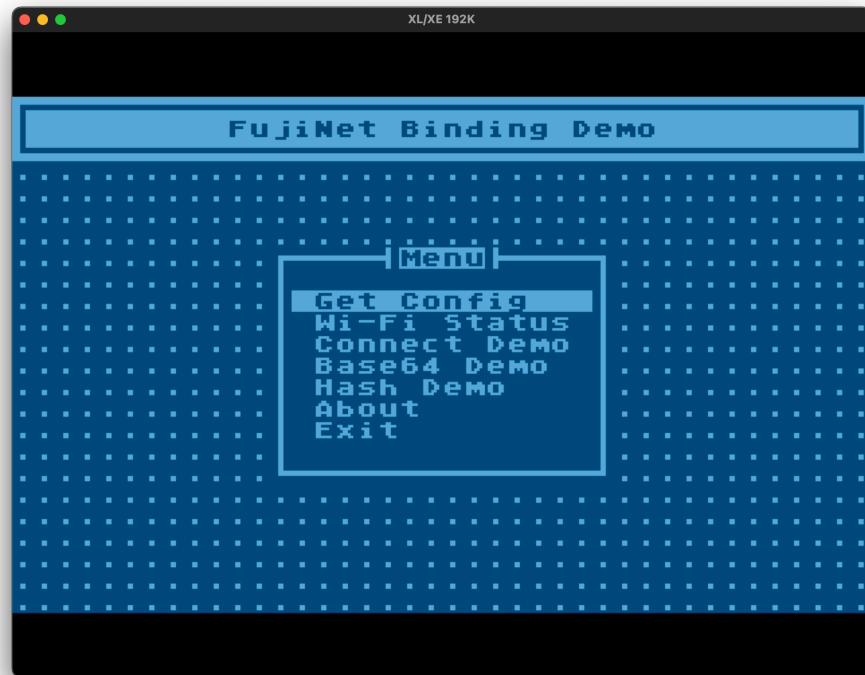
```



---

## Demonstration FujiNet Application

This is a demonstration of using the TUI (text user interface) library along with the FujiNet library. It shows how to include the libraries when building an application that uses both parts of the library.



File: FUJIDEMO.C

```
// -----  
// Program: fujidemo.c  
// Desc...: A8 FujiNet Binding Demo Application  
// Author.: Wade Ripkowski  
// Date...: 20230825  
// License: GNU General Public License v3.0  
// Notes..: cl65 -v [-O] -t atari fujidemo.c -o fujidemo.com  
// -----  
  
// Pull in include files  
#include <stdio.h>  
#include <conio.h>  
#include <unistd.h>  
#include <string.h>  
#include <stdlib.h>  
#include <fcntl.h>  
#include <atari.h>  
  
#include "a8defines.h"  
#include "a8defwin.h"  
#include "a8defsio.h"  
#include "a8libmisc.c"  
#include "a8libstr.c"
```

```

#include "a8libwin.c"
#include "a8libgadg.c"
#include "a8libmenu.c"
#include "a8libfuji.c"
#include "a8libfujib64.c"
#include "a8libfujihash.c"

// Prototypes
void About(void);
void TestCfg(void);
void TestCon(void);
void TestB64(void);
void TestHash(void);

// Globals
static unsigned char cU[256] = "N:HTTP://68k.news:80/";

// -----
// Func...: void About(void)
// Desc...: About Dialog
// -----
void About(void)
{
    byte bW;

    // Show window
    bW = WOpen(1, 6, 38, 10, WOFF);
    WOrn(bW, WPTOP, WPLFT, "About");
    WPrint(bW, WPCNT, 1, WOFF, "FujiNet Binding Demo");
    WPrint(bW, WPCNT, 2, WOFF, "Version 1.00");
    WPrint(bW, WPCNT, 3, WOFF, "(C) 2023 Wade Ripkowski");
    WPrint(bW, WPCNT, 5, WOFF, "A8 C-Library FujiNet");
    WPrint(bW, WPCNT, 6, WOFF, "binding demonstration.");
    WPrint(bW, WPCNT, 8, WON, " Ok ");

    // Wait for key
    WaitKCX(WOFF);

    // Close window
    WClose(bW);
}

// -----
// Func...: void TestCon(void)
// Desc...: Open, get and Close test.
// -----
void TestCfg(void)
{
    td_fnCFG sC;
    unsigned char bS, cL[33];
    byte bW;

```

```

// Clear memory
memset(cL, 0, 32);

// Get config into structure
bS = FNGConfig(&sC);

// Check status and report
if (bS != FNSOK) {
    GAlert("Connect: Failed to get config!");
} else {
    // Open window
    bW = WOpen(3, 5, 34, 17, WOFF);
    WOrn(bW, WPTOP, WPLFT, "Configuration");
    WPrint(bW, 1, 2, WON, "Version");
    WPrint(bW, 1, 5, WON, "Hostname");
    WPrint(bW, 1, 8, WON, "IP");
    WPrint(bW, 1, 11, WON, "SSID");

    // Fill in values from the returned structure
    // Version
    WPrint(bW, 1, 3, WOFF, sC.version);

    // Hostname, copy up to 32 chars for window constraints
    strncpy(cL, sC.hostname, 32);
    WPrint(bW, 1, 6, WOFF, cL);

    // IP address, clear the line buffer and build new string from the bytes
    memset(cL, 0, 32);
    sprintf(cL, "%03d.%03d.%03d.%03d", sC.ip[0], sC.ip[1], sC.ip[2], sC.ip[3]);
    WPrint(bW, 1, 9, WOFF, cL);

    // SSID, clear the line buffer, copy up to 32 chars for window constraint
    memset(cL, 0, 32);
    strncpy(cL, sC.ssid, 32);
    WPrint(bW, 1, 12, WOFF, cL);

    // OK "button"
    WPrint(bW, WPCNT, 14, WON, " Ok ");

    // Wait for key
    WaitKCX(WOFF);

    // Close window
    WClose(bW);
}
}

// -----
// Func...: void TestCon(void)
// Desc...: Open, get and Close test.
// -----
void TestCon(void)
{
    byte bW;

```

```

unsigned char *cB;
unsigned char cL[37];
unsigned char bS = 0, bL = 0;

// Get some memory and clear it
cB = malloc(768);
memset(cB, 0, 767);
memset(cL, 0, 36);

// Open window
bW = WOpen(1, 1, 38, 22, WOFF);
WOrn(bW, WPTOP, WPLFT, "Connect Demo");
WPrint(bW, 1, 1, WOFF, "URL:");
WPrint(bW, 6, 1, WOFF, cU);
WDiv(bW, 2, WON);
WPrint(bW, WPCNT, 2, WON, " Page Source ");
WDiv(bW, 19, WON);
WPrint(bW, WPCNT, 20, WON, " Ok ");

// Open connection
bS = FNOpen(cU);

// Set translation mode
FNTrans(FNTRLF, FNAXUPDATE);

// Check status and report
if (bS != FNSOK) {
    GAlert("Connect: Failed to open!");
}

// Get data
bS = FNRead(598, cB);

// Check status and report
if (bS != FNSOK) {
    GAlert("Connect: Failed to read!");
}

// Close connection
bS = FNClose();

// Check status and report
if (bS != FNSOK) {
    GAlert("Connect: Failed to close!");
}

// Display data
for (bL = 0; bL < 16; bL++) {
    strncpy(cL, cB + (bL * 36), 36);
    WPrint(bW, 1, 3 + bL, WOFF, cL);
}

// Wait for key
WaitKCX(WOFF);

// Close window

```

```

WClose(bW);

// Free memory
free(cB);
}

// -----
// Func...: void TestB64(void)
// Desc...: Base64 test (encodes then decodes).
// -----
void TestB64(void)
{
    unsigned char *cS = "InverseATASCII";
    unsigned long iS;
    unsigned char cB[FNHASHLENHSHA5 + 1], cL[33];
    byte bW;

    // Clear memory
    memset(cB, 0, FNHASHLENHSHA5);
    memset(cL, 0, 32);

    // Open output window
    bW = WOpen(3, 8, 34, 9, WOFF);
    WOrn(bW, WPTOP, WPLFT, "Base64 Demo");
    WPrint(bW, 1, 1, WOFF, "Text:");
    WPrint(bW, 7, 1, WOFF, cS);
    WDiv(bW, 2, WON);
    WPrint(bW, WPCNT, 2, WON, " Encoded ");
    WDiv(bW, 4, WON);
    WPrint(bW, WPCNT, 4, WON, " Decoded ");
    WDiv(bW, 6, WON);
    WPrint(bW, WPCNT, 7, WON, " Ok ");

    // ----- Encode the string -----
    // Push the unencoded string to the FN input buffer
    FNB64Inp(FNLENCODE, cS, strlen(cS));

    // Perform the encode
    FNB64Cmp(FNLENCODE);

    // Get the length of the result
    FNB64Len(FNLENCODE, &iS);

    // Get the encoded string from FN
    FNB64Out(FNLENCODE, cB, iS);

    // Show encoded
    WPrint(bW, 1, 3, WOFF, cB);

    // ----- Decode the string -----
    // Push the encoded string to the FN input buffer
    FNB64Inp(FNLDECODE, cB, iS);

    // Perform the decode
    FNB64Cmp(FNLDECODE);
}

```

```

// Get the length of the result
FNB64Len(FNLDECODE, &iS);

// Clear buffer before getting result
memset(cB, 0, 128);

// Get decoded string from FN
FNB64Out(FNLDECODE, cB, iS);

// Show decoded
WPrint(bW, 1, 5, WOFF, cB);

// Wait for key
WaitKCX(WOFF);

// Close window
WClose(bW);
}

// -----
// Func...: void TestHash(void)
// Desc...: Hash test (SHA1).
// -----
void TestHash(void)
{
    unsigned char *cS = "InverseATASCII";
    unsigned char cB[165], cL[33];
    byte bW;

    // Clear memory
    memset(cB, 0, 164);
    memset(cL, 0, 32);

    // Open output window
    bW = WOpen(3, 4, 34, 16, WOFF);
    WOrn(bW, WPTOP, WPLFT, "Hash Demo");
    WPrint(bW, 1, 1, WOFF, "Text:");
    WPrint(bW, 7, 1, WOFF, cS);
    WDiv(bW, 2, WON);
    WPrint(bW, WPCNT, 2, WON, " SHA1 ");
    WDiv(bW, 5, WON);
    WPrint(bW, WPCNT, 5, WON, " SHA256 ");
    WDiv(bW, 8, WON);
    WPrint(bW, WPCNT, 8, WON, " SHA512 ");
    WDiv(bW, 13, WON);
    WPrint(bW, WPCNT, 14, WON, " Ok ");

    // ----- SHA1 -----
    // Push the string to the FN input buffer
    FNHashInp(cS, strlen(cS));

    // Perform the hash
    FNHashCmp(FNHASHSHA1);

```

```

// Get the hash string from FN as hex
FNHashOut(FNHASHSHA1, FNHASHOHEX, cB);

// Show hash across 2 lines
strncpy(cL, cB, 20);
WPrint(bW, 1, 3, WOFF, cL);
strncpy(cL, cB+20, 20);
WPrint(bW, 1, 4, WOFF, cL);

// Clear memory
memset(cB, 0, FNHASHLENHSHA5);

// ----- SHA256 -----
// Push the string to the FN input buffer
FNHashInp(cS, strlen(cS));

// Perform the hash
FNHashCmp(FNHASHSHA2);

// Get the hash string from FN as hex
FNHashOut(FNHASHSHA2, FNHASHOHEX, cB);

// Show hash across 2 lines
strncpy(cL, cB, 32);
WPrint(bW, 1, 6, WOFF, cL);
strncpy(cL, cB+32, 32);
WPrint(bW, 1, 7, WOFF, cL);

// Clear memory
memset(cB, 0, FNHASHLENHSHA5);

// ----- SHA512 -----
// Push the string to the FN input buffer
FNHashInp(cS, strlen(cS));

// Perform the hash
FNHashCmp(FNHASHSHA5);

// Get the hash string from FN as hex
FNHashOut(FNHASHSHA5, FNHASHOHEX, cB);

// Show hash across 4 lines
strncpy(cL, cB, 32);
WPrint(bW, 1, 9, WOFF, cL);
strncpy(cL, cB+32, 32);
WPrint(bW, 1, 10, WOFF, cL);
strncpy(cL, cB+64, 32);
WPrint(bW, 1, 11, WOFF, cL);
strncpy(cL, cB+96, 32);
WPrint(bW, 1, 12, WOFF, cL);

// Wait for key
WaitKCX(WOFF);

// Close window
WClose(bW);

```

```

}

// -----
// Func...: void main(void)
// Desc...: Main routine
// -----
void main(void)
{
    // Variables
    byte bW1, bW2;
    byte bC = 1, bD = FALSE;
    unsigned char *pcM[8] =
    { " Get Config ", " Wi-Fi Status ", " Connect Demo ", " Base64 Demo ", " Hash Demo ", "
About ", " Exit " };
    unsigned char bS = 0;

    // Setup screen
    WInit();

    // Set Background
    WBack(14);

    // Open header window
    bW1 = WOpen(0, 0, 40, 3, WON);
    WPrint(bW1, WPCNT, 1, WOFF, "FujiNet Binding Demo");

    // Open menu window
    bW2 = WOpen(12, 7, 16, 11, WOFF);
    WOrn(bW2, WPTOP, WPCNT, "Menu");

    // Loop until done (Exit selected)
    while (! bD) {
        // Call menu
        bC = MenuV(bW2, 1, 2, WOFF, bC, 7, pcM);

        // Process choice
        switch (bC) {
            // Get config
            case 1: TestCfg();
                    break;

            // Get device status
            case 2: bS = FNStatus();
                    switch (bS) {
                        case FNDSSIDLE: GAlert("FujiNet Idle");
                                    break;
                        case FNDSSNOSSID: GAlert("FujiNet No SSID");
                                    break;
                        case FNDSSCANC: GAlert("FujiNet Scan Complete");
                                    break;
                        case FNDSSACTIVE: GAlert("FujiNet Active");
                                    break;
                        case FNDSLCFAIL: GAlert("FujiNet Last Connect Failed");
                                    break;
                        case FNDSWIFILOST: GAlert("FujiNet WiFi Connect Lost");

```



```

                                break;
                                case FNDWIFIDISC: GAlert("FujiNet WiFi Disconnected");
                                break;
                                }
                                break;

// Open and close test
case 3: TestCon();
      break;

// Base64 test
case 4: TestB64();
      break;

// Hash test
case 5: TestHash();
      break;

// About box
case 6: About();
      break;

// Exit
case 7: bD = TRUE;
      break;
}

// Exit on ESC as well
if (bC == XESC) {
    bD = TRUE;
}
}

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

// Set Background
WBack(0);

// Exit
return;
}

```