

# **eXtended AES Library 1.00**

Ken Hollis/Bitgate Software

COLLABORATORS			
	TITLE : eXtended AES Library 1.00		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY	Ken Hollis/Bitgate Software	February 14, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>eXtended AES Library 1.00</b>	<b>1</b>
1.1	XAES Version 1.00 . . . . .	1
1.2	XAES Programming Guidelines . . . . .	1
1.3	definitions . . . . .	2
1.4	callbacks . . . . .	2
1.5	main program callbacks . . . . .	2

## Chapter 1

# eXtended AES Library 1.00

### 1.1 XAES Version 1.00

XAES Version 1.00 by Ken Hollis of Bitgate Software

1. Guidelines
2. Library definitions

### 1.2 XAES Programming Guidelines

XAES Programming Guidelines

XAES is a self-contained GUI programming library. XAES is not a replacement in any way for AES, OS, GEM, or VDI. The guidelines for creating routines are as follows:

1. In any GEM program, if your program uses a resource file, the resource file should be loaded first (`rsrc_load`.)
  2. After which, the mouse should be changed to an arrow state (either by using `WGrafMouse`, or `graf_mouse`.)
  3. Resource objects should then be found, and saved in memory in your main program (using `rsrc_gaddr`.)
  4. Objects should then be fixed if they have any Extended Object types at this point and time, with `fix_objects`.
  5. `WInit` should be called.
  6. Attach any code to any objects you wish to attach code to (using `WAttachCode`.)
  7. `menu_bar` should then be called if you have a menu bar to display.
  8. Call `WDoDial` after all initialization has taken place.
  9. After the `WDoDial` routine, `unfix_object` should be called to remove any
-

Extended Object type routines attached to objects.

10. WTerm should be called.

11. Any error codes should be returned at this point.

Follow these guidelines and you will create a perfect XAES program.

## 1.3 definitions

XAES 1.00 Library

XAES's intrinsic library is one of the most complex intrinsic libraries on the Atari, yet it is among the easiest to use. XAES's intrinsic libraries were designed around the idea of X/Motif, and many ideas and simplicities were taken from those programming environments.

XAES basically has two types of application intrinsics:

- Initialization intrinsic
- Termination intrinsic

The initialization intrinsic is the basis for initialization of your XAES library program. This intrinsic function simply opens the GEM desktop, VDI workspace, and initializes NKCC.

The termination intrinsic simply deinitializes XAES, closes all windows that remain on the screen, closes the desktop (if applicable), closes the VDI workspace, and deinitializes NKCC.

## 1.4 callbacks

Method -- Callbacks

Callbacks are used mainly for handling the calling of routines in a simplified manner. XAES uses callbacks to call routines whenever an action takes place (like the click of a button, or whatever.)

There are different types of callback routines. Click on the one you wish to get more information on:

- Main program callbacks
- Window routine callbacks

## 1.5 main program callbacks

Method -- Callbacks -> Main program

There are currently four callbacks that are used for main program processing. WInit handles the calling of these callbacks depending on whether or not

---

you tell it to. If you don't want XAES to handle the callbacks automatically, you can control callbacks yourself.

The four callback indices:

```
XC_INITIALIZE  (1)  -- Initialization callback type
XC_DEINITIALIZE (2)  -- Deinitialization callback type
XC_STARTUP     (3)  -- Startup callback type
XC_EXIT        (4)  -- Exit callback type
```

Setting callbacks:

To set callbacks, the routines are quite simple. Each routine you want the callback to point must have no calling parameters. A sample callback routine would be written as:

```
LOCAL void MyInitCallback(void)
```

As you can see, all callbacks for the main program must have no calls. They may contain calls to other functions within the callback, but the callback itself cannot contain any parameters.

To place the callback into memory, the function to call is:

```
WSetCallback(type, rout);
```

where TYPE is the callback index type, and ROUT is a pointer to the routine. Thus, if I wanted to make MyInitCallback the INITIALIZATION callback, I would call the routine thusly:

```
WSetCallback(XC_INITIALIZE, MyInitCallback);
```

Calling your callback manually:

To call any main program callback, the method is as follows:

```
DoCallback(type);
```

where TYPE is the callback index type.

Macros have been set to make things easier for you. The following macros point to the following indices (this was taken straight from the header file):

```
XCallInitializeCallback  - DoCallback(XC_INITIALIZE);
XCallDeinitializeCallback - DoCallback(XC_DEINITIALIZE);
XCallStartupCallback     - DoCallback(XC_STARTUP);
XCallExitCallback        - DoCallback(XC_EXIT);
```

These should be used instead, but of course, you can use the DoCallback routine. That's what it's there for.

---