GemFast v1.8

Public Domain GEM Programming Library
By Ian Lepore

Installation and Portability Notes

Contents

Packing List

GemFast v1.8 is distributed in 3 archive files, as follows:

GEMFST18.LZH -	Libraries, headers, and documentation.
GEMFSC18.LZH -	Library source code.
GEMFXM18.LZH -	Example application source code.

The GEMFST18.LZH archive contains the following files:

AESFAST.A -	The AES library.
VDIFAST.A -	The VDI library.
GEMFAST.H -	GEM programming header file.
EXTERROR.H -	Extended error message header file.
GEMFAST.DOC -	The library reference manual.
GEMFINST.DOC -	This document.
AESVRSN.DOC -	Revision history for AES library.
VDIVRSN.DOC -	Revision history for VDI library.
LD.TTP -	New linker, required for HSC users.

The GEMFSC18.LZH archive contains two other archives, which in turn contain many files. (It compresses much tighter this way.) When unpacked, the following directory structure emerges:

VDIBIND\ -	All VDI library functions.
AESBIND\ -	Low-level AES bindings.
AESXBIND.S\ -	Extended bindings (ASM code).
AESXBIND.C\ -	Extended bindings (C code).
AESUTIL.S\ -	Low-level utilities (ASM code).
AESUTIL.C\ -	Low-level utilities (C code).
AESFUNCS\ -	The bulk of the high-level code.
INCLUDE\ -	Header files.

The GEMFXM18.LZH archive contains the following files:

(need a list here)

Installation and Usage Instructions

These installation instructions assume you are using GemFast with a DRI-compatible compiler, and are using GemFast as your primary GEM bindings system. Compatible compilers are:

Sozobon C v1.x and v2.0
Heat and Serve C (HSC) v1.34 or higher
Laser / MegaMax C
Alcyon C

The distributed libraries are NOT compatible with other compilers. If your compiler is not on the above list, please proceed to the Portability Notes chapter.

Basic installation

Installation of GemFast is fairly simple. Copy the VDIFAST.A and AESFAST.A files to the directory your compiler uses for libraries. If you use ALN or another indexed linker, be sure to run DOINDEX to build new library index files. Copy the GEMFAST.H and EXTERROR.H files to the directory your compiler uses for header files. Put the documentation wherever you want.

LD.TTP Installation (for HSC users)

If you're using Heat-and-Serve C v1.3x, you need to copy LD.TTP to replace your current version. Older versions of LD had some compiled-in limits on the number of object files in a library. There are now so many modules in the GemFast libraries that these limits get exceeded. The new LD has larger default limits, and also some new command line options that let you increase the limits for any given run. The new command line options are:

-lo#### The number of object files referenced/used.
-lm#### The number of object files skipped/unused.
-ls#### The number of non-local symbols.

The default limits are 300 used object modules, 400 unused modules from libraries (modules skipped during multipass processing), and 1000 non-local symbols. To double these limits, add the following to your LD command line:

-lo600 -lm800 -ls2000

The numbers you specify must be between 1 and 32767 inclusive. The larger the numbers, the more memory is needed at runtime.

Usage

To link your programs with the GemFast libraries, just include the library names on the linker command line. Example:

```
cc myprog.c aesfast.a vdifast.a dlibs.a
```

There is now a linker order dependancy! Many of the new high-level functions require VDI services, so you must now place VDIFAST.A after AESFAST.A on your linker command line. The libraries are still internally sequenced so that your linker will only need to make one pass of each library. Also, some of the new high-level dialogs require runtime library services such as sprintf(), so it is best if the GemFast libraries appear before your C runtime library.

You can use any runtime startup module you want, when your compiler/linker provide several startup modules. Nothing in the GemFast libraries relies on items in the startup files.

IMPORTANT RECOMPILE WARNING!

You MUST recompile all your existing GEM object modules as a part of installing v1.8! Because macros in GEMFAST.H remap some standard GEM calls through new internal library routines, it is imperative that all your GEM-related object code be recompiled using the new GEMFAST.H header file before you link *anything* with the new libraries.

It is hard for me to overstate the importance of this. Everything will die horribly if you link an existing object module containing GEM calls with the new libraries.

Here's my suggestion: Go look *everywhere* on all your partitions or disks. Nuke all object files that have even a remote chance of containing a GEM call. If you have your own libraries that contain GEM calls, nuke the libraries. Do all the deletions at once, then go through and recompile things where necessary.

GemFast System Overview

GemFast provides complete support for GEM programming. It provides the GEM support for the Sozobon and HSC public domain compilers. In addition, it replaces the existing GEM support for the Laser, MegaMax, and Alcyon compilers. The low-level bindings and utilties are written in hand-coded assembler to provide maximum speed and memory efficiency. The higher level functions in the AES library are written in C to provide portability to other compilers.

The GEMFAST.H header file

The GEMFAST.H header file defines constants and datatypes frequently used in GEM programming. In addition, it contains declarations for all functions which return a value other than an int.

This header file also contains macros which remap old utility function names to the newer (supported) names, and macros which route standard GEM calls such as graf_mouse() through new support routines such as grf_mouse(). The remapping is done at a macro level to provide compatibility with compilers not using the low-level GemFast bindings. This means that you MUST include GEMFAST.H into every source code module that contains GEM function calls.

This header file replaces the OBDEFS.H and GEMDEFS.H files delivered with other GEM support systems.

The AESFAST.A library

The AESFAST.A library contains the low-level bindings to standard AES functions, low-level utilities such as rectangle calculations, and the high-level library functions. All standard AES functions are supported, including fsel_exinput(), and other functions Atari added to the AES in TOS 1.04. (Special code in the fsel_exinput() binding makes it work properly on all TOS versions.)

The GEMFAST.DOC file documents all functions in the AESFAST.A library which are not standard AES functions. (Refer to any standard GEM reference manual for descriptions of the standard AES functions.)

The VDIFAST.A library

The VDIFAST.A library contains low-level bindings to the standard VDI functions. It does not contain every VDI function documented in standard GEM programming manuals, but it contains all the important ones. (A few rarely-used functions such as v_cellarray() and polaroid palette driver functions are misssing.) It does not contain any GDOS or FSMGDOS functions. (I expect to get docs on GDOS soon, then I can add bindings for them.)

Stack and memory usage

The low-level bindings use no data memory other than the global variables documented in GEMFAST.DOC. Most low-level bindings use 50 bytes or less of stack space. A noteable exception is v_gtext(), which uses 50 bytes plus two times the length of the string you pass to it.

The higher level functions in the AES library use more stack space, but rarely more than a couple hundred bytes, and never more than 1k. (I've used the new dynamic dialogs with a 2k stack without any problem.)

The dynamic dialogs contain embedded resource data, and this has bloated the AESFAST.A library (and will do the same to your program) a bit. Where possible, I've made similar functions share internal data structures such as embedded resource data to try to keep the total data size of your programs down to a reasonable size.

GEM and supervisor mode

You cannot make AES calls from supervisor mode; it has to do with the way the AES internals save registers. With some herculean efforts you can hack around it if you need to, but it'll require custom assembler code on your part.

You can make VDI calls from supervisor mode.

If you use G_USERDEF objects, be aware that the custom drawing routine you supply will be called in supervisor mode. This means that it cannot make AES calls. Also, if you make DOS, BIOS, or XBIOS calls from the custom drawing routine, you may get a spurious stack overflow error message from your C runtime library bindings (gemdos(), bios(), and xbios() functions), because being on the supervisor stack will confuse any stack checking these routines do.

Portability Notes

Starting with GemFast v1.8, all high-level library functions are coded in C, and should be fairly portable to other compilers. I've yet to try porting the libraries, and I'd expect a few problems on a compiler where sizeof(int) is 4 bytes.

The extended bindings (evnx_multi(), etc) and the low-level utilities (rc_intersect(), etc) exist in both C and assembler source code form. For use with MWC or another compiler that doesn't use DRI-standard object modules, just compile the code in the AESXBIND.C and AESUTIL.C directories with your C compiler, and they will take the place of the assembler routines from AESXBIND.S and AESUTIL.S. The C functions won't be as small or fast as the assembler versions, but they will let you port programs written for GemFast to your compiler.

I hope to expand on these portability notes in future releases, as well as making whatever code changes are needed to increase portability. I welcome feedback from anyone who ports GemFast to other compilers.

Note added at a much later date: I have now done a preliminary port of GemFast to Lattice C 5.52. The good news is that a release of GemFast that is compatible with both ANSI and K&R compilers is a real possibility in the near future. The bad news is that it may be a commercial product rather than freeware. However, if it does become a commercial product, I promise that it will be very reasonably priced, and that if you've sent me a donation on the current version, you will receive a discount on the commercial version if you choose to upgrade.

Ian Lepore
05/31/92 (updated 12/06/92)
BIX ianl
Internet ilepore@nyx.cs.du.edu