GemFast v1.9

Public Domain GEM Programming Library
By Ian Lepore

Installation, Usage, and Portability Notes

Contents

Distribution Archives

GemFast now supports many compilers, and is distributed in compiler-specific archives. The following files should be available at your favorite public domain software outlet. In the names, 'vv' refers to the version number.

GFvvSRC.LZH	-	Complete source code.
GFvvXMP.LZH	-	Example application source code.
GFvvDOC.LZH	-	Documentation (reference manual).
GFvvHSC.LZH	-	Heat and Serve C v2.x libraries.
GFvvLC5.LZH	-	Lattice C v5.x libraries.
GFvvGCC.LZH	-	GNU C v2.x libraries.
GFvvTPC.LZH	-	Turbo C / Pure C libraries.

The source code archive contains complete source code for both low-level bindings and high-level library functions, plus associated header files, makefiles, etc.

The example archive contains several example applications written using GemFast features.

The documentation archive contains the function reference manual and this installation document. Sorry to put the docs in a separate archive, but they're awfully big these days, and some folks will be needing to download several of the compiler-specific archives, and it would be nasty to have to download duplicates of all those documents each time.

The compiler-specific archives contain the libraries and header files needed to use GemFast with that compiler. Detailed contents of these archives appear in the installation sections for each compiler, below.

HSC Installation and Usage Instructions

Versions of GemFast beginning with v1.9 are designed for use with HSC v2.0 or higher. GemFast is no longer compatible with HSC v1.x, or Sozobon C (unmodified), because it now uses the long-external-names feature of HSC v2.0. The object modules in the library are in standard GST format, and should work with any tools which support the HSC stack-parameters calling standard and GST-format object modules.

GemFast for HSC provides both a set of low-level bindings to the OS GEM routines, and a library of high-level functions to simplify GEM programming.

Archive contents

The GFvvHSC archive contains these files:

GEMFAST.A	-	The library.
GEMFAST.H	-	The main header file.
GEMFBIND.H	-	The low-level bindings header file.
EXTERROR.H	-	Header file for extended error messages.

Installation

Installation of GemFast is fairly simple. Copy the GEMFAST.A file 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, GEMFBIND.H, and EXTERROR.H files to the directory your compiler uses for header files. Put the documentation wherever you want.

When acquired with the HSC distribution, the HSC installation program has already installed GemFast for you, but please read the following sections.

Usage

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

```
cc myprog.c gemfast.a dlibs.a
```

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.

Some of the GemFast high-level functions contain references to standard runtime library functions, such as sprintf(). The GEMFAST.A library name should appear before your runtime library name on the linker command line.

GNU C Installation and Usage Instructions

GemFast v1.9 can only be used with GNU C when using the -mshort (16-bit integers) option. The next release will support 32-bit integers as well. If you forget the -mshort option, the GEMFAST.H header file will abort the compile with a #error directive reminding you that this version works only with the -mshort option.

GemFast for GNU C provides both a set of low-level bindings to the OS GEM routines, and a library of high-level functions to simplify GEM programming.

When used with GNU C, all low-level and high-level functions are declared with ANSI-style prototypes in the header files.

Archive contents

The GFvvGCC archive contains these files:

GEMFST16.OLB	-	The 16-bit library.
GEMFAST.H	-	The main header file.
GEMFBIND.H	-	The low-level bindings header file.
EXTERROR.H	-	Header file for extended error messages.

Installation

Installation of GemFast is fairly simple. Copy the GEMFST16.OLB file to the directory your compiler uses for libraries (any directory in your GNULIB= list). Copy the GEMFAST.H, GEMFBIND.H, and EXTERROR.H files to the directory your compiler uses for header files (any directory in your INCLUDE= list). Put the documentation wherever you want.

If you have the old GEM support system for GNU C, you can delete the libraries and header files that came with it, since GemFast completely replaces it.

Usage

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

```
cc myprog.c gemfst16.olb -lgnu16
```

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.

Some of the GemFast high-level functions contain references to standard runtime library functions, such as sprintf(). The GEMFST16.OLB library name should appear before your runtime library name on the linker command line.

Lattice C Installation and Usage Instructions

GemFast for Lattice C has been compiled using Lattice C v5.52. Whether it will work with pre-5.5 versions of LC is anyone's guess at this point.

GemFast for LC supports both 16- and 32-bit integers, and both near and far (base-relative and absolute addressing) data models. The libraries are compiled using near code model, so you'll have to have ALV linking enabled to use the library.

GemFast v1.9 for LC doesn't support the register parameter passing options of LC right now. This will be rectified in the next release.

GemFast for LC provides only the high-level functions documented in GEMFAST.DOC. The standard LC bindings libraries and header files are used for low-level GEM support. This means that, unlike other compilers, you specify the GemFast library in addition to the compiler's GEM library, not instead of it.

When used with LC, all low-level and high-level functions are declared with ANSI-style prototypes in the header files.

Archive contents

The GFvvLC5 archive contains these files:

GEMF.LIB	-	Long integer base-relative model lib.
GEMFNB.LIB	-	Long integer non-base-relative model lib.
GEMFS.LIB	-	Short integer base-relative model lib.
GEMFSNB.LIB	-	Short integer non-base-relative model lib.
GEMFAST.H	-	The main header file.
EXTERROR.H	-	Header file for extended error messages.

Note that the suffixes on the GEMFxxxx.LIB files correspond exactly to those used by the LC runtime libraries. If you are using LCSNB.LIB as the runtime library for a project, then you would use GEMFSNB.LIB as the GemFast library.

Installation

Installation of GemFast is fairly simple. Copy the .LIB files to the directory your compiler uses for libraries (any directory in your LIB= list). Copy the GEMFAST.H and EXTERROR.H files to the directory your compiler uses for header files (any directory in your INCLUDE= list). Put the documentation wherever you want.

Usage

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

```
lcc -w myprog.c gemfs.lib lcgs.lib
```

If you are using the Lattice IDE and its project manager, just do a Project|Edit|Add and add the apporiate .LIB file name along with all your source code module names in the project window. Be sure to include the path name for the library in the IDE's Options|Environment|LIB dialog, if you put the GemFast libraries in a different path from the compiler's libraries.

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.

Some of the GemFast high-level functions contain references to standard runtime library functions, such as sprintf(). The GEMFxxxx.LIB library name should appear before your runtime library name on the linker command line.

Turbo C and Pure C Installation and Usage Instructions

(guess I'll know this after I con someone into compiling up the Turbo and Pure C versions of the library for me.)

MWC No Longer Supported

GemFast no longer supports MWC. It has, quite simply, outgrown the capabilities of the MWC compiler. In fact, since HSC v2.0 now supports long external names, I would recommend that anyone still using MWC consider switching to HSC (or perhaps GNU C) instead. Or best of all, get Lattice C, if you can afford it.

GemFast System Overview

GemFast provides complete support for GEM programming. It provides the basic GEM support for the HSC and GNU C freeware compilers. For other compilers, the high level functions use the existing compiler bindings. The low-level bindings and utilties are written in hand-coded assembler to provide maximum speed and memory efficiency. The higher level functions are written in C to provide portability to other compilers.

Differences from prior GemFast versions

Starting with GemFast v1.9, the GemFast library is contained in a single file, GEMFAST.A, which replaces the two files VDIFAST.A and AESFAST.A used in prior versions. There is also now an additional header file, GEMFBIND.H.

You will need to change any existing makefiles which contain the two old library names, subsituting the new name GEMFAST.A. (If you're using PAMAKE, set USESGEM=1 in your makefile and let the MAKE.INI file set up gemfast in the linker command for you.)

When using HSC and upgrading from v1.8 or earlier to post-v1.8 GemFast, you will need to recompile all your existing object modules and libraries. This is due to the new HSC 2.0 support for long external names; an existing module will have, for example, a reference to function graf_sli, and the function in the library is now known to the linker by its full name, graf_slidebox.

The GEMFAST.H header file

The GEMFAST.H header file defines constants and datatypes needed to use the high-level portion of the GemFast library. In addition, it contains prototypes for all functions in the high-level library. It also contains some conditional sections which help make GemFast compatible with all existing ST C compilers.

Part of the conditional logic in GEMFAST.H causes the proper set of header files for the compiler's GEM bindings to be included. For some compilers, this will be GEMFBIND.H, described below. For other compilers, it will be the file(s) delivered by the compiler vendor for AES and VDI bindings. For maximal portability of your source code, you should #include <gemfast.h> in your application modules, and let GEMFAST.H include the proper low-level headers for the compiler in use.

The GEMFAST.H 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.

If you are using GemFast for the low-level bindings but you are not using any of the functions or variables from the high-level part of the library, you can #define NO_GEMFAST_HLL before your #include <gemfast.h>. This will eliminate the remapping macros that redirect appl_init(), appl_exit(), and other functions through the high-level support routines. This gives you, in effect, a set of fast low-level bindings that are 100% compatible with the DRI standard, but avoids the extra overhead of the high-level library's advanced features. If you use NO_GEMFAST_HLL, you must be very careful not to use any of the functions or variables declared in GEMFAST.H, because they won't work correctly without the special init and cleanup that normally gets done when the appl_init/exit macros route your calls through the high-level library.

The GEMFBIND.H header file

For the HSC and GNU C compilers, GemFast provides the low-level GEM bindings support for the compiler. The GEMFBIND.H header file contains function prototypes, macros, and datatypes commonly used in GEM programming. All low-level GEM bindings conform to the DRI/Atari documentation of those functions. (EG, evnt_timer() takes a pair of unsigned short words, not a long word like some alternate bindings have defined.)

This header file is roughly analogous to the OBDEFS.H and GEMDEFS.H files from the old Alcyon compiler, or the AES.H and VDI.H files from Lattice C and other newer compilers.

Both VDI and AES items appear in GEMFBIND.H for simplicity in maintaining GemFast. If you aren't using VDI and you want to gain a tiny measure of performance, you can #define NO_VDI before including GEMFAST.H, and the VDI-related items will be skipped in GEMFBIND.H. Likewise, #define NO_AES can be used to elimated AES-related items.

The GEMFAST library

The GEMFAST library (which exists under varying names for some compilers, to support multiple memory models) contains the entire GemFast implementation. For HSC and GNU C, that includes both high-level library functions and low-level OS bindings. For other compilers, it contains just the high-level functions, and is used along with the compiler's bindings libraries.

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() and similar VDI string input/output functions, which use 50 bytes plus two times the length of the string you pass.

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 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.

Using G_USERDEF drawing functions

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.

Also, if you are using a compiler with a base-relative memory model, be aware that there can be problems with accessing data a G_USERDEF drawing routine. If you declare the function using the GCALLBACK type, your compiler will generate code to load the data base register on entry to the routine, eliminating this problem. (The GCALLBACK type is defined in GEMFAST.H).

Portability Notes

GemFast is now a fairly portable library. The low-level bindings for 16-bit compilers are still written in assembler for maximum efficiency. For use with GNU C in -mshort mode, the HSC JAS assembler is used with the -u (unix a.out format output) option to compile the low-level bindings. This avoids the need for a second copy of the low-level source in MIT assembler syntax.

The high-level code is written in fairly portable C, in a way that seems (so far at least) to work just fine with both Classic C and ANSI C compilers.

The high-level code does assume that the low-level bindings use 'short' as the basic datatype for the GEM interface. You will have nothing but trouble trying to port this code to a 32-bit compiler that uses 'int' as the GEM datatype.

Ian Lepore
04/23/93
BIX   ianl@bix.com   (preferred email address)
Internet ilepore@nyx.cs.du.edu