Default

Paul Manias

COLLABORATORS			
	TITLE : Default		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Paul Manias	July 26, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Default

1.1 The Games Master System V0.9B

T H E G A M E S M A S T E R S Y S T E M

BY PAUL MANIAS

VERSION 0.9B

GENERAL DOCUMENTATION

1.2 Introduction to the Games Master System

This introductory text is taken directly from the web pages:

http://gms.ethos.co.nz/gms/

WHAT IS THE GAMES MASTER SYSTEM?

The Games Master System is a solution to one of the biggest problems the Amiga community has ever faced. What problem? Games support in the OS! Windows has it, the Apple has it, and the Amiga has none. This lack of support has brought up some quite serious problems when it comes to hardware compatibility in games. Every Amiga owner encounters them at some time, if not often. There have been efforts made towards libraries specifically tailored to handling graphics cards, but none of them address some of the most important aspects of games programming, such as why direct hardware access is done in the first place.

Hardware access is fast, often many times as fast as an equivalent system legal routine. However direct hardware access is not very compatible with other hardware types, resulting in unpredictable results on other systems. Many existing games and demo writers have been reluctant to move to using libraries, because they don't offer the necessary speed. To combat this you have to upgrade all of your hardware to something faster - but it only avoids the problem, not solve it. It has already happened to the PC, forcing users to pay large amounts of money for over-powered systems.

The problem also goes deeper than just graphics. How about sound? User support? Networking? Joysticks, JoyPads, Analogue joysticks, 3 Button mice? Adequate debugging facilities? Where is the real support for writing games in an OS? Could we ever expect to support have all of this and more, while still keeping the environment fast? Well after almost two years of work, that answer is yes.

THE GAMES MASTER SYSTEM

GMS solves all of the problems that have just been outlined and beyond. It is a fully grown development system, providing comprehensive documentation, developer information, lots of source code, a user prefences program, an official system debugger and more. All these things have been designed to work together, and the following goals have become a reality:

* Erradication of the need to bash the hardware from within games.
* Easier to migrate from the current Amigas to the new Power (or whatever) Amigas.
* It is fully portable to any platform (PC version due by 1999).
* Makes games programming, easier, faster, and more productive.
* Gives users the ability to modify any program to suit their requirements.

On top of all this, GMS also features resource tracking, object orientation, CPU assisted drawing, data protection, and loads more. If you want to see how easy GMS is to program, just look at the source code available on these web pages.

Lastly, one of the most important features of the Games Master System is the preferences program. This allows you to select levels of mode promotion, type of joystick used, C2P routines, task priorities, screen properties, and so on. This solves a lot of the moans and gripes that

users have had in the past, and since this is all transparent to the programmer, user support is easily achieved. Hopefully this news will make you all very happy!

ANYTHING ELSE?

Look around the rest of the site to get more detail on the things mentioned here, and other things that we haven't gone into yet. The GMS binaries and documents are available on Aminet, in dev/misc/gms_dev.lha and dev/misc/gms_user.lha. Remember to ask us if you have any questions about the project!

Return To Index

1.3 OverView

OVERVIEW OF THE GAMES MASTER SYSTEM

Project GMS started in the beginning of April 1996, in an effort to provide games support in the Amiga OS. The overall aim is to write the best games interface we possibly can, which should eventuate into a system that everyone can enjoy. Although the development of GMS is largely controlled by myself (Paul Manias) I would like people to see it as a project of the Amiga community and I am completely open to ideas and comments. The current objectives of the project are:

1. To erradicate the need to bash the hardware from within games.
2. To make it easier to migrate from the current Amigas to the new Power Amigas.
3. To make games programming easier, faster, and more productive.
4. To give users the ability to modify any program to suit their requirements.

GMS has been designed to be fully extendible in ways that will make future improvements very easy to implement. The system is split into a number of sub-sections: The kernel, the modules, the debugger, and the preferences program. This is further enhanced by identifiable data objects, which allow us to write enhance system objects in the future, without overhauling the functions. GMS has no problems with future compatibility, since hard-coded structure definitions are disallowed and tag-lists are very well supported.

Looking to the Future
In 1998 I will release all of the module source code to the public domain. The only thing that I will continue to develop is the Kernel, Preferences, and System Debugger. I will also continue to define all of the standards, include files, documentation and so on. In short, I will be moving into more of a system management role.

This means that somone else will have to write the modules, and that

someone is you. With the source freely available, I am hoping that people will begin to develop support for graphics and sound cards, enhanced modules, fixing bugs and so on. I have ended up developing a project that is finally too big for me to handle alone, and I think this is the best way to get everyone involved. The PPC version is entirely dependent on you, because I will not be converting the modules.

Please help me all that you can, because I cannot do it without you.

Return To Index

1.4 Questions and Answers

QUESTIONS AND ANSWERS

I often get mail from people asking me questions about what you can and can't do in GMS. Here I will answer some of these questions, and hopefully this way everyone can benefit in learning more about how GMS works. If you have any questions you can mail them to me and they may appear here.

GENERAL

Q. Will you support new machines such as those from PIOS and the upcoming OS's like pOS?

A. Buy the 680x0 version, then I can quit my job and put more effort into this kind of support. Failing that, when the source code is released I hope that developers will pool their efforts and write hardware and interface drivers for any machine that you want.

Q. Is it possible to free structures that have not been initialised? [The idea being that it makes it easier to write the initialisation code]

A. Yes, a standard feature of GMS is to recognise structures that have not been initialised. Many functions check if you have passed them null or invalid structures, so the security in this area is very solid.

BLITTING

Q. If I want to have 5 Bobs with the same graphics, may I initialise the first with NULL in Bob->MaskData and GENMASK set, then copy the pointer created to the other structures and init them with GENMASK cleared? Otherwise the masks would be created again several times and waste memory.

A. Yes, it is legal to copy masks generated in one Bob over to another Bob. Just remember when you free the "master" bob, all Bob's containing pointers to its masks will become invalid. For this reason make sure that you free the master bob last.

Q. How does CPU assisted blitting work when parallel drawing means that

there could be an instance of data overlap (CPU and blitter draw to the same area at the same point in time)?

A. It's a clever trick... What you do is start the blitter drawing the first 5 lines or so of the bob. While it does that you use the CPU to start drawing 5 lines from the bottom of the bob. When the CPU finishes with its section it checks on the blitter. If the blitter has finished then the CPU sets it blitting the next 5 lines and the CPU continues on. This keeps going until the blit is finished.

Q. I have successfully initialised a Bob with a Picture attachment. when I draw it the dimensions are correct but the graphic is corrupt.

A. Make sure that you have specified the MEM_VIDEO or MEM_BLIT flags in the MemType field of the Picture's Bitmap. If you forget to set one of these flags then the blitter will probably be attempting to blit from fast memory, resulting in corrupt graphics.

Return To Index

1.5 How to write new modules.

WRITING NEW MODULES

Anyone can write a module, but there are a couple of things you have to do first. You must decide what kind of module you are going to write, understand object orientation under the DPKernel, look at the source to other modules and read the module guidelines. If you don't know about child classes, hidden objects, or system classes, go back and read up on this before continuing!

A module can do basically two things: It can contain system objects (eg the blitter module carries the Bob and MBob objects), and it can contain functions that perform certain actions (eg DrawPixel()). The system modules are pretty much focussed around objects, but you may want to write one that consists entirely of functions. On the other hand, the JoyPorts and Picture modules are entirely object orientated, leaving the kernel to support the functionality of their objects.

An example of writing a new module
Let's say you wanted to add jpeg support to the Picture class. Because you will be adding support to a class that already exists, you will be creating a child class module. The first thing to do in this case is write to paul@ethos.co.nz and ask if such a module has already been written. You may change your mind if it's already been done. If everything is well you will be sent back some information on what to call your module (in this case, probably "mod.jpeg") and if necessary, a unique module identifier (not required for child modules).

The next thing to do is write an object referencing file. You can view a list of reference files in System/References/. You'll notice that these

files are extremely small, so obviously it won't take long to write one. Our particular reference file will end up looking like this:

```
  INCDIR  "INCLUDES:"
  INCLUDE "dpkernel/dpkernel.i"

  SECTION "Reference",DATA

Start:  dc.l  TAGS_REFERENCE,0
        dc.l  REFA_ObjectID,ID_HIDDEN
        dc.l  REFA_ObjectName,.name
        dc.l  REFA_ModName,.module
        dc.l  REFA_CheckFile,.checkfile
        dc.l  TAGEND

.name   dc.b  "Picture-Jpeg",0
        even

.module dc.b  "mod.jpeg",0
        even

.checkfile
  cmp.l   #$FFD8FFE0,(a1)
  bne.s   .chk0
  cmp.l   #"JFIF",6(a1)
  bne.s   .chk1
.chk2 move.l  #99,d0
  rts

.chk1 move.l  #60,d0
  rts

.chk0 move.l  #00,d0
  rts
```

[Notice that although ID_HIDDEN was specified as the ObjectID, this only concerns referencing. The actual Jpeg object itself will be initialised as a child object later, not a hidden object.]

At this point I'm going to stop, I'll come back and write some more later :-)

Conditions of Module Development

Modules are grouped into two different types: Class modules, which carry parent classes and functions, and Support modules, which may carry functions, hidden objects and child classes. 80% of module programmers should fit into the second category, for which there are minimal requirements. Class modules are developed under fairly strict guidelines because they are much more important. Here are the conditions:

Class Modules

If you intend to write a module that will contain code for a parent class, you MUST:

1. Register the module and object(s) by writing to paul@ethos.co.nz. [You will receive the necessary ID's to start development].

2. Write accurate and comprehensive documentation for the object and module over their continued development.

3. Four weeks before you intend to release the first version, you must send the module and any relevant information to paul@ethos.co.nz to get final approval of object and function definitions. Alternatively you may show what you're doing on a regular basis to keep everything on track.

4. If in the event that you stop writing your module you should pass all of the development information (ie source code) to a person of your choice. Alternatively you can send it to DreamWorld Productions so that we can find someone that wants to continue its support. We don't like to see modules created and then dropped without continued development!

If you do not follow the above guidelines, DreamWorld Productions will not give your work any official recognition what so ever. We also reserve the right to directly prevent your module from running even if it has been correctly installed. Remember there may be a lot of people using your module, so we must ensure that it's 100% OK and can be upgraded for the future.

Support Modules
If you are writing a module containing a set of functions, and/or 1 or more hidden objects or child classes, you should:

1. Register the name of the module by writing to paul@ethos.co.nz. [This is done simply to prevent naming conflicts in the System/ directory.]

Return To Index

1.6 Really cool features!

CURRENT FEATURES OF THE GAMES MASTER SYSTEM

This is just a summary of the major features that have so far been implemented. Not all new features and changes have been documented here. For the complete low-down on all features of GMS check the developer information files.

* Completely object oriented system design, covering all aspects of OO including data/function inheritance and polymorphism. This allows for much more powerful programming, data abstraction and modularity.

* Multiple platform capabilities. A GMS program compiled on a 680x0 Amiga could also be run on a 680x0 Atari or Mac, all you would need is the necesary drivers. All PPC compilations will have the same feature and can also be 680x0 compatible through emulation.

* Resource tracking is fully implemented, a task can exit and all its resources will be completely freed. A SelfDestruct() function allows a task to abort itself at any time and the system will free its

resources - extremely useful for debugging purposes. A user may force a task to abort simply by pressing L-AMIGA and DELETE, 100% safe.

* Debugging support implemented into all initialisation functions, no need for unnecessary patches to get debug information or track system calls. A debugger utility exists for receiving and displaying this data in real time.

* Transparent Chunky-To-Planar, which means it won't bother wasting time with conversions or copying if chunky mode is already available in the hardware.

* Fast blitter functions for drawing bob's, copying for screen buffers, 3 different screen clears, auto background saving and clearing for bob's. Also includes Pixel and Line drawing functions, and support for list's for very fast mass-drawing operations. CPU assisted blitting means that drawing speed is no longer limited to blitter throughput.

* Sound support includes: Support for sound priorities, intelligent dynamic channel play-back, channel modulation for special effects, IFF support.

* Proportional colour fading, functions are: PaletteMorph(), ColourMorph(), PaletteToColour() and ColourToPalette(). Support for setting speed and colour ranges.

* Full support for raster/copperlists, with effects such as: ColourLists, Mirror, Flood, and Palette Changes.

* Structure and object pre-processing, allows data to be changed separately from the main program. This makes GMS the first system to support up to 100% user editing of game data.

* Allows you to support all different kinds of input devices (joysticks, joypads, mouse etc) through just one simple function call. This enables you to support devices that don't even exist yet.

* User preferences program to allow full configuration of a game's functionality. This includes configuration for: Game/Task Priorities, Choice of networking, Mode Promotion, Joystick Config, Music Redirection, and more.

* Stable memory allocation and a freemem routine that will not crash your machine if you have written over your memory boundaries. Internal resource tracking ensures that GMS programs will not chew up your memory.

* Smart Saving and Loading of files, with automatic packing and depacking via XPK.

* 320k of assembler, E and C sources, demonstrating all uses of the library.

* All GMS programs can multi-task with no significant drop in speed or performance.

Return To Index

1.7 What language to use?

LANGUAGES

As GMS is no more than an extension to the OS, it can work with any language that you want it to. Currently supported languages in this archive are:

Assembler
C/C++
E

You could also use Blitz Basic, Pascal, Oberon and others if you know how, but I currently don't have any source demos or special include files to help you with those.

If you're new to programming then I would recommend starting out with C or E. In my opinion E is a little easier on the beginner, but C is more common place in other areas and you might find that useful. If you don't know which one to choose, try learning E first, and then C as they are quite similar languages. Unfortunately E has some portability problems, so if you intend to do things properly you will need to upgrade to C at a later date.

If you want to write really fast games, you will have to learn some assembler. With GMS learning assembler is quite easy, as you don't have to think about programming the hardware registers. Look at the demo sources and make up your own mind if you want to learn it or not. Using GMS you could become a fairly adequate assembly programmer in as little as 2 months if you have come from something like C. Bear in mind that assembler makes programs difficult to port.

WHAT COMPILER?

If you know what language you want to use, you will have to think about what compilers you should get. You can't program without a compiler! Here is my opinion on the most common and best compilers available:

Assemblers
The best assemblers are AsmOne, DevPac and PhxAss. I have all three of these and use each one of them for different situations. You don't need that many, but two of these are free, so it won't cost you anything.

AsmOne has an excellent source-level debugger and I recommend it to beginners, as you can observe how the 680x0 instructions work. I don't use it that much today, but it is useful and has some features that make it very easy to use. It also has the fastest compiler speeds that you could imagine. I got the latest AsmOne from the WWW, go to one of the Amiga Web searches and look for "AsmOne" to find it.

DevPac is a good, robust compiler with many options, but it's a little slow and hasn't been updated in a while. I recently moved to using PhxAss for these reasons. You can get DevPac from HiSoft and other software dealers, it's a commercial product so you will have to pay for it.

I have been using PhxAss for a while and have found it to be a very impressive asm compiler. It is compatible with DevPac sources and has very good compiling times. The package is regularly upgraded and it's freeware. Good work Frank Wille! You can get PhxAss from Aminet, just download it as dev/asm/Phx*.lha.

You will also need a text editor if you're using DevPac or PhxAss, I recommend CygnusEd as it's small and you can alter the TAB stops. This feature is important as it keeps your sources easier to write and manage. You will notice that all my assembler sources look strange unless you view them with CED or AsmOne. CygnusEd has been upgraded recently, so now is a good time to buy.

C Compilers
SAS C/C++ is what I use most often, it's very reliable and I've never had a problem with it. The documentation is very extensive, so you'll be able to get help for all your problems. This product is no longer officially supported, but you can order the remaining copies on the WWW. You may be able to get it from various software dealers. Dice C is a nice package but it doesn't support any C++. It has recently been released as freeware, so it's worth getting as your first C compiler. You can get this one from Aminet, in dev/c/ I believe.

Again, you will need a good text editor for efficient programming. CygnusEd is ideal here, and I believe GoldEd is a popular choice as well.

E Compilers
There is only one E compiler available (EC) which you can get as part of the E package. You can get this from Aminet, along with everything else that you will need. You will probably have to register, although this program was put on a coverdisk some time ago.

Return To Index

1.8 Hints and Tips

GAMES MASTER SYSTEM

HINTS AND TIPS

This section is written to offer some friendly advice and tips on how to get full use from the Games Master System, and what tricks you can use to make sure your game runs at the highest speed possible. I'm still writing this section, but if you have a trick of your own that should be here, please write to me at paul@ethos.co.nz. Even though I wrote the system, I

don’t know everything that can be done with it :-)

1.1 GENERAL CODING TIPS

Less... equals More!

Never call the same routine twice in your main loop unless absolutely necessary. For example, look at this routine that calls Query() twice:

```
----
Loop: move.l  DPKBase(pc),a6
      move.l  KeyStruct(pc),a0
      CALL    Query
      move.l  KeyStruct(pc),a0
      move.l  KEY_Buffer(a0),a0
      cmp.b   #K_ESC,(a0)
      beq     Game_Over

      ...
      Rest of main loop
      ...

      move.l  KeyStruct(pc),a0
      CALL    Query
      move.l  KeyStruct(pc),a0
      move.l  KEY_Buffer(a0),a0
      cmp.b   #" ",(a0)
      beq     .Exit
      ...
      bra.s  Loop

KeyStruct:
      dc.l   0


----
Do this instead...


Loop: move.l  DPKBase(pc),a6
      move.l  KeyStruct(pc),a0
      CALL    Query
      move.l  KeyStruct(pc),a0
      move.l  KEY_Buffer(a0),a0
      cmp.b   #K_ESC,(a0)
      beq     Game_Over

      ...
      Rest of main loop
      ...

      move.l  KeyStruct(pc),a0
      move.l  KEY_Buffer(a0),a0
      cmp.b   #" ",(a0)
      beq     .Exit
      ...
```

```
      bra.s   Loop
```

As you can see the second version is faster because it doesn't make an extra call to Query(). Simple really, but it often happens to beginners and in large programs.

1.2 CONTINUATION OF TASK PROCESSING WHILE PAUSED

There are times when pausing of your main task (through WaitAVBL()) will be inconvenient if it is necessary to continually process information. On the other hand, if your program continues to run in the background it will steal the processor for as long as it continues drawing.

Lets say you are writing a game that can connect via the serial port for 2 player communications. If one machine was to stop its processing, the serial buffer will continue to receive information and could go into overflow, potentially causing you some problems when your task is reactivated. The easy solution to this is to activate a secondary task that will continue to process when the main task is paused. This is a simple procedure and only requires that you put all your communication handling into this separate task. Another method is to use an interrupt, although that is not necessary in this case.

An option that may be more convenient for the user in a TCP environment, would be to send out a message saying "This machine is temporarily paused" so that all other machines know that they must not send information to you. This will give any other TCP tasks running on the paused machine more time to send/receive data, eg for FTP.

1.3 SUPPORTING HIGHER RESOLUTIONS

Drawing high resolution graphics and supporting them as an option in your game is a worthwhile exercise, and will make the owners of more powerful computers happy. But it can be annoying to support - most developers make two copies of each picture file, one in lo-res and one in hi-res and then program the game to support both files. This can get in the way of programming the game itself and results in wasted time. In GMS there is a way to solve this problem.

Draw all the graphics in high resolution and use them as you normally would in your game. Use screen tag lists that accept the default screen dimensions from the user (do not set GSA_ScrWidth, GSA_ScrHeight or GSA_ScrMode).

Set the RESIZE flag when loading in the pictures and set PIC_Width and PIC_Height in accordance to the user's resolution in the screen that you opened.

Example: If the game graphics were drawn on a 640x512 screen and the user has asked for a LORES screen, then drop the picture dimension to 320x256 and load it in. The picture will be resized to fit the new dimensions and you now have the lo-res equivalent of your hi-res screen.

The next step is to proportionalise your bobs to the new settings. There

are two fields to help you do this - PropWidth and PropHeight. These fields must contain the original dimensions of the Bob's source picture, which in the example above was 640x512.

Now, when you call Init(), the function will detect that the Width of the source Picture does not match the PropWidth setting (same thing for the height). It will then use a formula to alter your Bob's Width, Height and coordinates to reflect the new dimensions.

```
Bob->Width = (Picture->Width) / (Bob->PropWidth / Bob->Width)
```

That's it. There are some proportional demos in this archive, check those to see how easy the procedure is (note how there is no extra coding needed, just the addition of a few tags and the RESIZE attribute).

Return To Index

1.9 Compatibility Problems

COMPATIBILITY PROBLEMS

One of the most important decisions I made in the design of GMS, was to get the absolute most out of what the Amiga hardware is capable of. The fact is, if I wrote GMS with respect to other gfx cards, there would be no:

Sprites
Hardware Scrolling
Overscan
Double Playfields
Split Screens
and Raster lists

Strangely enough, isn't this what makes the Amiga unique? Also if the new Amiga's came out with quadruple 256 colour playfields and 512 colour sprites, should I support that if other gfx boards don't? Why should I not support it?

Well one of the goals of GMS, is to always be as up to date as the hardware that is available at the time.

Now, this is at the cost of compabitility. How compatible you want your game to be on other systems is entirely your own choice. Generally, the more hardware-specific features you use, the more you risk your games failure on different hardware. If you use no special effects, your game has an excellent chance of successfully working on all systems. Whatever happens, you will have to make your own decisions on the compatibility issue.

What I will do in the future of this section is try and help you face these problems, and hopefully overcome them. With some intelligent programming, you can still use features like sprites, rasters and hardware scrolling, and still keep your game running on other systems. It takes a little work

but the least it will do is make a lot of people very happy. Good luck!

Return To Index

1.10 The Authors

THE AUTHORS

The Games Master System is written in Assembler and C by Paul Manias. Paul has 5 years 680x0 and games programming experience, and another 2 years in other languages like C and Pascal. Paul's favourite past-times are blowing his nose, staring at the ceiling, and lurking in basements. So far he has written two games of his own and contributed graphics to two other commercial ones. None of those games have beeen released (yet?), for all sorts of various reasons. Luckily this is not the case with GMS.

GMSPrefs was orginally written in E, by Richard Clark. Richard's favourite past-times are standing, sending morse code via blinking, and talking to suspicious items of furniture.

Many thanks to Graeme Chiu, who hosted the GMS pages from April 1996 - May 1997. To see our WWW pages, visit:

http://gms.ethos.co.nz/gms/

Thanks to Jyrki Saarinen and Fabio Bizzetti for their donations to the project. Some of the C demos were converted by Adam Dawes, French documentation was written by Julien Boibessot, and Gerardo Iula has come up with some good ideas while using GMS. Thanks to the people that send in useful bug reports, and to the many people that sent in ideas when the project first started (but we still need more!).

There is a questionnaire that I would like you to fill out and return to us. So far I have questionnaires from the following people:

Dag Agren, Daniel Rost, David Johanson, Fabio Rotondo, Frank Hauptlorenz, Julian Boibessot, Ken Axsom, Mark Papadakis, Martin Kuchinka, Miguel Ramos, Mikael Drugge, Porter Woodward, Robbie Bankston.

Not enough! So if your name isn't there, start writing :-)

Return To Index

1.11 Resource Tracking

RESOURCE TRACKING

GMS is fully supportive of internal resource tracking, which means that it tracks resources without any effort from the programmer. Resource tracking is great for programming as it warns you if you have forgotten to free important system allocations when your program exits. This is not just memory, but also things like sound, blitter, video display, files and device allocations. This becomes a life saver in situations such as forgetting to free a hardware allocation like the blitter, as this would normally cause a system deadlock and you would have to reset your machine. Fortunately resource tracking will rescue a situation like this and you can get the system back with everything intact.

TASK DESTRUCTION

Resource tracking also gives us the opportunity to use an even greater feature, which is task destruction. Task destruction is the ability to stop and destroy a task while it is performing its normal processes, and still leave the system intact. You can stop a program immediately by holding LEFT-AMIGA and DELETE - even in the middle of a video game! This powerful feature uses resource tracking to return your system to the same state that it was in before the program was active. This is quite handy for users and programmers that want to get back to their system as quckly as possible, and obviously is very useful for debugging purposes.

Return To Index

1.12 Copyright Notice

COPYRIGHT

Return To Index