How to write portable code for Amiga
(AmigaOS 3.x / MorphOS / AmigaOS4 / AROS)

By Mathias PARNAUDEAU - mathias.p@wanadoo.fr
12-feb-2006

To help developers to follow the evolution of the last years, this document aims to give information and advices to develop using a standard way where the code will be compiled on our various Amiga systems : AmigaOS 3.x, MorphOS, AmigaOS 4, AROS.

The AmigaOS API is almost the same and some libraries are now integrated : AHI for the audio management, CGFX (supported by Picasso 96), SDL, TTEngine, ... But for the developer, there are sometimes big differences between ReActiion and MUI, Poseidon and Sirion, ...

Now each OS adds new functions or mecanisms like AmigaInput under OS4 for the input device management or Reggae under MorphOS for the multimedia.

Before to see those differences about API, includes and libraries, let me give some general advices :
- realize your project with a modular goal in mind : even if a confuding code gives a working program, it will be difficult to improve or debug it, mainly for external developers
- encapsulate the major differences like the access to devices like USB ports : don’t call directly specific functions.
- respect the langage conventions and avoid special tricks of the compilers (the best way is to use several compilers)
- use the advantages of the operating systems and their tools (splint, Grim Reaper, GDB, mpatrol, ...), portability doesn’t only mean constraints !
- use several compilers to show errors and warnings, each one has its own tolerance

The document describes more precisely the differences between SDK and how to manage a unique environment with common rules to write and maitain portable code purchasing a minimum of effort.

We consider here that the reader already knows the basics of programming and has experience of the development on Amiga with the C langage.

Last, you will find a list of resources at the end of this document.

Note : This is a quick english version done by me, so if my english is too bad, please send me corrections. If the technical content shows some mistakes, let me know too.

1. Includes

Includes contain what the system gives to the developer : structures, constantes, function declarations, ... Some includes are compiler or system dependant : pragma, inline, ppcinline, ... They don’t have to be called directly from a program. We have to include only proto files that will call the right include files.

1.1. OS4, a specific case

Under OS4, the basetype of the libraries (declared in the proto files) are now almost all turned to “struct Library” instead of the library name with the Base suffix (like IntuitionBase) or particular cases (SysBase pour Exec, GfxBase pour Graphics, ...). Depending on the OS revision, the types have changed sometimes and this adds some complexity. For a question of compatibility and if we want to keep the original types (Library or XxxBase), we have to compile with the define __USE_BASETYPE__.

The inline files are put in the directory “inline4” and thanks to their macros, they do the link between the function declarations and their interfaces, these new things bring with OS4. We talk about interfaces in a next part.

1.2. includes generation

The main and historical tool is called fd2pragma and is able to generate each type of includes for a library from its description file, FD.
For MorphOS, a Perl script nammed cvinclude.pl has the same role. Another script (UpdateMosIncForVbcc.pl) to patch the includes for a better support of VBCC (see at http://developer.morphosppc.com/index.php, in the part “ Files “).
With OS4, fd2pragma is only used to create a XML file, a kind of super FD file which allow a new tool called idltool to create the includes and library skeletons too.

1.3. Includes organisation with several compilers

When we develop with several OS and/or several compilers, the management of the include directories can give a true headache. Some includes depends on the compiler and have nothing to see with the AmigaOS API.

Other includes are common to all the compilers like SDL, OpenGL, MUI classes, ... It’s better to keep them in a unique place to give at the compiler, in its configuration or with the include path option (-I for GCC and VBCC, IDIR for SAS/C). Under OS4, the SDK organisation already manage a such directory which the path “ SDK:Local/Common/Includes “.

Be careful, these common includes are often associated to static libraries that depends on the OS but their format can depends on the compiler too ! VBCC manages this giving different targets in which we have 2 directories : includes and lib.

2. OS specificities

Make its code portable still requires to look at some particularities. We tell how to do here.

2.1. Specific defines

The OS or compilers particularities will be seen below but for some specific parts, there are some defines to allow conditional compilation of the code :

OS4 : __amigaos4__
MorphOS : __MORPHOS__
AROS : __AROS__

VBCC : __VBCC__
GCC : __GNUC__
SASC : __SASC

Be careful : To chose which define to apply, it is necessary to have a global point of view on the project and include or exclude particular cases. For example, we want to port an audio application from AmigaOS 3.x to MorphOS and we want to use the hardware in the first case and AHI in the other one. For this last case, we won't use "#ifdef __MORPHOS__" but something like "#ifdef USE_AHI". So, even if it's a MorphOS port, the code will be ready to be compiled under OS4.

2.2. OS4 interfaces

This is a new concept that allows to specify explicitely the library of a used function.Even if its mecanism support several interfaces for one library, another interest is the clear code with a kind of C++ syntax, for example : IExec->OpenLibrary(...).

To write portable code, if the interface advantages are not mandatory, that's better to leave this mecanism. In appearance only because interfaces are still there ! To keep the classic function calls without any changes in the code, we have to compile with the option -D__USE_INLINE__. Under OS4 we still need to declare interfaces that are used. With the option -lauto, the main libraries (exec, dos, intuition, ...) are opened and their interfaces initialized too (beware with VBCC).

To manage an interface, for example here with MUI, we declare it at the beginning of the source :

```
Struct MUIMasterIFace *IMUIMaster;
```

Then the interface is obtained in the initialization part of the program :

```
#ifdef __amigaos4__
IMUIMaster = (struct MUIMasterIFace *)GetInterface(MUIMasterBase,
“main”, 1, NULL);
#endif
```

... and this interface is removed at the end :

```
#ifdef __amigaos4__
if (IMUIMaster) {
   DropInterface((struct Interface *)IMUIMaster);
}
#endif
```

2.3. Functions amiga.lib, DoSuperNew

From the beginning, amiga.lib has became bigger with more functions but different ones depending on the compiler and the system. It is useful but doesn't represent a common base and it can change again in a near future.

The best example of evolution comes from OS4 in which some functions were moved out to be put in their logical library. For example, DoMethod went to Intuition with the name of IDoMethod. This is why it's necessary to use this define under OS4 : -DDoMethod=IDoMethod

La function DoSuperNew is a special case. It exists in the MorphOS amiga.lib but misses with OS4 or AmigaOS 3.x (SAS/C, VBCC, GCC). It is used to create a new object, for example in the constructor of a MUI custom class.

There are several ways to define this functions but no one offers the guarantee to work eberywhere ! The simplest code I found (using includes from SDI headers, see below) looks like that :

```
#ifndef __MORPHOS__
#ifdef __amigaos4__
Object * STDARGS VARARGS68K DoSuperNew(struct IClass *cl, APTR obj, ...)
{
   Object *rc;
   VA_LIST args;

   VA_START(args, obj);
   rc = (Object *)DoSuperMethod(cl, obj, OM_NEW, VA_ARG(args, ULONG),
NULL);
   VA_END(args);

   return rc;
}
#else
APTR STDARGS DoSuperNew(struct IClass *cl, APTR obj, ULONG tag1, ...)
{
   return ((APTR)DoSuperMethod(cl, obj, OM_NEW, &tag1, NULL));
}
#endif
#endif
```

Remark : Under MorphOS, it seems that if we link with aboxstubs o we use -DUSE_INLINE_STDARG, this declares DoSuperNew with this macro :

```
#define DoSuperNew(MyClass, MyObject, ...) \
    ({ ULONG _tags[] = { __VA_ARGS__ }; \
    struct opSet MyopSet; \
    MyopSet.MethodID = OM_NEW; \
    MyopSet.ops_AttrList = (struct TagItem*) _tags; \
    MyopSet.ops_GInfo = NULL; \
    DoSuperMethodA((MyClass), (MyObject), (APTR) &MyopSet); \
    })
```

The keyword VARARG68K is not used here and that can help because it is not known by all the versions of GCC.

2.4. Task creation

With the jump to the PowerPC processor, some things have to be adapted due to the hardware architecture. This is the case of tasks that were well known too for their missing features. Under OS4 there is no changes to apply but under MorphOS we have to add a tag in the CreateNewProc call :

```
#ifdef __MORPHOS__
    NP_CodeType,    MACHINE_PPC,
#endif
```

MorphOS offers other new features like the creation and automatic freeing of the message port. You can send user data thanks to the tag NP_UserData (this feature is available in AROS too).

2.5. Memory

The evolution of our systems raises one thing that has to be improved : the memory management. Memory allocations use flags (with the prefix MEMF_) to set their type. Under OS4, it is now forbidden to use MEMF_PUBLIC : if a special case requires to allocate a shared memory block, there is a new flag called MEMF_SHARED. In common programming, we just have to use MEMF_ANY or MEMF_PRIVATE.

MorphOS offers new flags too : MEMF_SWAP for example to manage the virtual memory using hard drives. This is certainly an equivalent to MEMF_VIRTUAL in OS4.

3. Compilers specificities

3.1. Macros from SDI_headers

At the beginning each compiler came with its own keywords (like reg, __asm, __saveds, ...), using its own syntax. This iswhy Dirk Stoecker started to create some macros grouped in the file “SDI_compiler.h”. With the help of Jens Langner, it has became a set of includes available in a package called SDI_headers :

- SDI_compiler : registers, paramters handling, ...
- SDI_hook : macros for hooks and dispatchers
- SDI_lib :
- SDI_stdarg : varargs, etc.

So, all the compiler dependencies are masked behind these macros.

3.2. The hook case

A hook is a mecanism to associate a function to an event, so the function will be executed without any manual calls. With the example of a CD player, a hook could be created on the ejection event : even with a manual ejection, the program can manage that refreshing the GUI with no song titles. Hooks are often used with MUI too.

The mecanism is really close to the 680x0 registers and can be difficult to manage by hand. Fortunately, the include SDI_hook gives a simple multi-OS solution :

```
HOOKPROTO(nom_de_la_fonction)
{
   code de la fonction
}

MakeHook(nom_du_hook, nom_de_la_fonction);
```

The hook is then called by the function CallHook (with “&name_of_the_hook”) or the MUI version MUIM_CallHook. MakeStaticHook is used instead of MakeHook when the hook doesn’t have to be seen from outside.

HookEntry (in amiga.lib) was a common way to manage hooks with AmigaOS and MorphOS but it is not there in OS4.

In the same model of hooks about BOOPSI and MUI, we have the dispatcher which the definition is now simpler :

```
DISPATCHERPROTO(nom_du_dispatcher)
{
   code du dispatcher : switch avec les méthodes de la classe
}
```

An associated macro is called ENTRY and is needed when the dispatcher is set in the creation of the custom class :

cl_disko = MUI_CreateCustomClass(NULL, MUIC_Group, NULL, sizeof(struct DisKoData), ENTRY(DisKoDispatcher));

3.3. Functions with variable argument number

The files from SDI_headers help again here ! To write functions like printf with a variable argument number, the file “SDI_stdarg.h” offers macros to encapsulate the specific keywords. Please look at the DoSuperNew example above.

Annexe 1 : Resources

Cmmon references

VBCC
Free C compiler available at http://sun.hasenbraten.de/vbcc/

MUI
Site ZeroHerofor MUI classes : http://www.zerohero.se/mui/
SDK MUI : mui38dev.lha available on Aminet (http://www.aminet.net)

CD Developer
Includes, documentation (Rom Kernel Manual, ...)

Rom Kernel Reference Manual
Reference books, complete but to be reserved to advanced programmers

Amiga programming in french
Guru-meditation : http://www.guru-meditation.net
AmigaOS programming guide : http://amigadev.free.fr/ProgrammationAmigaOS.pdf

CubicIDE
Development environment : http://www.dietmar-eilert.net/cubic/index.htm

And of course all the tools and examples at http://www.aminet.net

MorphOS

SDK MorphOS
Available on the developer website : http://developper.pegasosppc.com

AmigaOS 4

SDK OS4
To ne downloaded with the OS updates at : http://www.hyperion-entertainment.biz

BitByBit
Development tools : http://bitbybitsoftwaregroup.com/index.php

OS4 depot
Other tools and examples at http://os4depot.net

AROS

Official AROS website
Information on AROS, development manuals, ... : http://www.aros.org

Support site for users and developers
News, forums, ISO images, ... sur http://www.aros-exec.org

Other links

http://www.monkeyhouse.eclipse.co.uk/amiga/dev.htm
http://www.utilitybase.com

And don’t forget to read the autodocs that are the first resources to get.