AFC_WhitePaper

COLLABORATORS			
	TITLE : AFC_WhitePaper		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 14, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AFC_WhitePaper

1.1 Amiga Foundation Classes

```
                Amiga Foundation Classes

A FreeWare and Public Domain library of Object Classes devoted to the
software development on the Amiga and on future Amiga compatible machines.

! Note: new / modified lines are signed with a "!" as the first char of the
!       line.
```

1. Index

1.2 What are the Amiga Foundation Classes

2.1 - What are the Amiga Foundation Classes

"Amiga Foundation Classes" (AFC from now on) is the name of a library of Object Classes designed to manage the several features of Amiga. Each Class will constitute an interface between the programmer and a particular aspect of the Amiga Operating System, trying to make the design and the coding of a program for the Amiga OS computers easier. The AFC have been also designed to assure a quicker porting of the Amiga programs on the future Amiga-Compatible Operating Systems. See
section 2.4
for more info.

1.3 What are the Amiga Foundation Classes for

2.2 - What are the Amiga Foundation Classes for

We have designed the AFC for different reasons: first of all the necessity of fully exploiting the object oriented programming (OOP), that allows to "recycle" code using it in other Classes or in other programs. The recompilation of the AFC on future computers with an Amiga-Compatible OS will allow a quicker porting of software products. See
section 2.4
for more
info.

1.4 Who will use the Amiga Foundation Classes

2.3 - Who will use the Amiga Foundation Classes

The AFC will be available to all programmers using a language supporting the OOP. The AFC can be used freely in FreeWare, ShareWare, PD or Commercial programs without paying any royalty.

```
! If you use the AFC, please, quote somewhere in the "About..." requester
! and also in the documentation that your program uses the Amiga Foundation
! Classes and provide also the official URL:
!
!        http://www.intercom.it/~fsoft/afc.html
!
! Note that you are just _invited_ to do so, no obbligation is given in
! any way.
```

1.5 Future of the Amiga Foundation Classes

2.4 - Future of the Amiga Foundation Classes

The AFC have been designed to warrant a reliable base of Objects to be used in the future Amiga-Compatible machines: by recompiling the AFC source code will be possible to obtain an excellent library of Classes with which one can easily build applications for the new computers. The advantage of using Classes of objects over the Operating System calls is due to the fact that such Classes work as an "interface" between the OS and the programmer: people that programmed on the Amiga using the AFC will bring their code on the new machines by simply recompiling the sources and using the AFC version implemented on the new computer on which they are working. Thank to the AFC, programmers will save a lot of time (otherwise spent in learning the new OS) and will keep on using the same Classes with the same sintax and the same behaviour. This implies, at least in our intentions, that the software will be developed better and in a shorter time. The risk that every new Operating System has to face, when it first comes to life, is the lack of professional software. Thank to the AFC it will be possible to provide in brief time professional products for these new OS.

1.6 Interfacing with new Amiga OS Compatibles

2.5 - Interfacing with new Amiga OS Compatibles

It’s our intention to freely supply developers of Amiga-Compatible OS with the source code of the AFC, be they Phase5, VisCorp, Be Inc., AROS, p-OS, PIOS, etc. Indeed, we are persuaded that the future of a new machine, possibly not depending on Microsoft, is to be found in the possibility of having in a short time software products of a professional quality. Thus the AFC sources will be provided to every OS developer that will ask for them. Then the particular implementation will be examined by the supervisors of AFC that will give their approval and allow the distribution. See section 5.0 for more infos.

1.7 How to write a Class for the Amiga Foundation Classes

```
                3.1 - How to write a Class for the Amiga Foundation Classes
```

1.8 Class name

```
3.1.1 - Class name

  - The Class must have a unique and understandable name.
  - The class name must be lowercase.
  - The class name must be in English.
  - The class name must NOT be the name of an Amiga system structure.

!   Please, note that the "_obj" naming convention has been discarted.
!   Use just unique names that don't collide against Amiga system structures
!   names


  - If in doubt, you can always add to the name of the class the suffix
    "_obj".
```

1.9 Methods' names

```
3.1.2 - Methods' names

  - The name of a method must be as shortest and clear as possible.  E.g.,
    in a class designed to manage a list of nodes the method that adds a
    node must NOT be called "addnode", but simply "add".
! - The name of a method must start in lowercase.

!   From V1.50 of AFC white paper, you can create names with UPPERCASE chars
!   to easy method naming:
!
!   ie. "addviewtocopperlist" now can become:
!       "addViewToCopperList" or similar.
!
!   please, note that the first chars *MUST* be lowercase anyway.
!
!   This way of naming is more close to Sun's Java language, another
!   place where we are planning to port AFC to.


  - The name of a method must be in English.
```

1.10 Object Inheritance

```
3.1.3 - Object Inheritance

The Object Inheritance tipical of the OOP must be used as little as you
can.  This because the Object Inheritance is implemented differently in the
various languages.  The AFC sources must be understandable to allow for an
easier porting and readability.  Thus it is FORBIDDEN the use of particular
syntax rules characteristic of a particular language.  E.g., are not
allowed structures of the "friend" type or the Multiple Inheritance of the
C++ language.

! It's advisable to substitute the Object Inheritance with the creation of a
! copy of the Class to be "reused" in the new Class.  E.g., if you have to
! inherit the Class "bitmapper" in the Class "picture" (notice the name
! lowercase and in English) it's advisable to copy the "bitmapper" Class in
! the "picture" Class.
```

1.11 Object Initialization and Object Destruction

```
3.1.4 - Object Initialization and Object Destruction

The initialization and destruction of a Class must be always performed by
means of two particular Methods:  the Initialization method and the
Destruction one.

The Initialization method must have the same name as the Class.  E.g., the
Initialization method of the "picture" Class must be called "picture()"
(with parameters between the brackets, if needed).

The Destruction method of a class must be named "end()" and must NOT
require any parameter.

! In the C++ version of AFC classes the destruction method can also
! be called with the standard C++ notation of the "~" char.
! So the Destruction method of the "picture" class in the example
! could also be: ~picture() with NO parameters, anyway.

These two methods must be called EXPLICITLY by the program that exploits
the Class:  you must not rely only on the Initialization and Destruction
routines contained in the language you use.

! If a Class uses another one, then it has to call in its own Initialization
! method the Initialization method of the other Class.  E.g., the "picture"
! Class that contains the "bitmapper" Class has to call, within the
! "picture()" method, the "bitmapper" method associated with the
! "bitmapper" Class.
```

1.12 Libraries and lowest OS version

3.1.5 - Libraries and lowest OS version

Classes have to rely almost exclusively on the standard AmigaOS libraries.

E.g., a "picture" Class that uses the "iff.library" won't be included in AFC because you can load an image also using the "iffparse.library" and then write the ByteRun-decoding routine.

Besides, the use of third party libraries damages the AFC, because these libraries might not be implemented on new Operating Systems.

The lowest OS version supported by AFC is 2.04. We can also take into account the compatibility with OS1.3, but it's not a fundamental feature.

If a Class requires a newer OS version to work correctly, then this must be documented.

1.13 Tags

3.1.6 - Tags

When possible the behaviour of the Class must be changed using tags, a characteristic of the AmigaOS. The standard methods involving tags must be called:

```
   - "setattrs(tags)" to set the tags.
   - "getattr(tag)" to get a single return value (with the corresponding
                    tag as parameter).
   - "getattrs(tags)" if you want to receive different values with the same
                      call.
```

1.14 Definition of error codes

3.2 - Definition of error codes

When to report an error code

How to report an error code

Classification of error codes

1.15 When to report an error code

3.2.1 - When to report an error code

The AFC must promptly react at all possible errors: they must be "idiot proof". If the user makes a mistake in calling a method, or he calls it before having initialized the Class, and so on, the Class must be able to trap and report the error (possibly trying to continue). Thus it's important to classify and report to the user the largest number of errors, trying to be understandable (when possible) to make the debug session easier.

1.16 How to report an error code

3.2.2 - How to report an error code

```
! To report an error it's preferable to use the (software) "Exceptions", when
! they are supported by the language; if they are not supported it's
! the called methdo can return the value of the exception as the return code.
! Anyway, it is _your_ task to provide a simple and good way to let the user
! handle exceptions and errors.
```

1.17 Classification of error codes

3.2.3 - Classification of error codes

The error code of a Class is contained in a 32Bit variable (long). The High word (2 bytes) contains the "identification number" of the Class: this identification number is unique and it's assigned by AFC to the Class when it is first created. This value must be stored in the source code of the Class in a constant named as the Class itself (written UPPERCASE) with the "_BASE" suffix. E.g., the picture Class has the identification value named "PICTURE_BASE". The Low word (2 bytes) contains the error code: in each Class it starts from $0000 and goes up to $FFFF (supposing that a Class can return 65535 different error messages!). The $0000 error it's always "No Memory". The table that associates to every error code its description must be included in the documentation.

1.18 Porting features

4.1 - Porting features

The AFC can (and must) be ported on the largest number of languages as possible. The porting must take into account all the characteristics listed in sections 2 and 3. It's forbidden to modify a Class during the porting, but the Class has to be "adapted" if some operations are not translatable into the language used (e.g., returning multiple values with a single method call).

1.19 Porting limits

```
4.2 - Porting limits

A Class ported by a programmer CANNOT be distributed before approval by the
AFC supervisors.  This to assure the correct working of the Class after the
porting.  We of AFC won't exclude any language, but the programmers that
won't accept these conditions won't receive any other Class to convert and
their Classes will be marked on the AFC WEB pages as "Not Supported".
```

1.20 Compiled Classes

```
5.1 - Compiled Classes

The compiled classes (e.g., the ".obj" files of C++ or the ".m" files of
AmigaE) may be freely distributed and have to be inserted in a directory
called "AFC" (obviously).

In the ".lha" archive must be present, together with the compiled code, the
documentation in AmigaGuide format and some example sources.
```

1.21 Classes' source code

```
5.2 - Classes' source code

The source of the Classes is NOT public domain an so it's NOT distributed
along with the compiled code.  Nevertheless, it will be always available to
programmers that ask us to do the porting of a particular Class.  The
supervisors of AFC will always have all the sources and are free to decide
whether to give the source of the required Class to a particular programmer
or not.
```

1.22 Availability

```
5.3 - Availability

The compiled versions of AFC can be found at these sites:

  FTP:
  ftp.intercom.it/pub/afc             - Official site of Amiga Foundation Classes

! HTTP:
!   www.intercom.it/~fsoft/afc.html     - Official WEB site of AFC


! The E version of the AFC classes will be also shipped with the language itself.
! But some of them will result to be unuseful without the registered version of
! the compiler. :-)
```

1.23 Info and WEB sites

```
5.4 - Info and WEB sites

If you want more infos (or upgrades) you can connect to:

  http://www.intercom.it/~fsoft/afc.html

   the official page of AFC.

Or you can subscribe to the mailing list by sending a message to:

  afc-list@intercom.it

with (in the SUBJECT): subscribe
```