```
ReadMe.txt for Tutorial/console
-------------------------------

+ ------------------------------+
| This is work in Progress      |
+ ------------------------------+

I haven't had time yet to write a tutorial about this matter.  However I know that Andr
e is working on a Framework and I think he'll find the information in this tutorial use
ful.

TODO:
More and better documentation!

Tutorial
--------

There are many ways in MacOSX to link applications and library code together.

In this application (console), I use the following methods:

1) direct compilation and linking
2) linking to a static library
3) linking to a dynamlic library
4) linking to a framework

Linking types 1 and 2 exist on every system.

Linking type 1 is almost trivial.  You compile all the code (both for the application a
nd the library code, and link it).  So in this case, you compile console.cpp and mylib.
cpp and they are linked to form console (an executable).

----

Linking type 2 is very link type 1.  However you build a library (by compiling mylib.cp
p).  All the compiled members of the library are collected into an ar file (an ancestor
 of todays tar) and called libmylib.lib.  When you compile the main program (console.cp
p), you link mylib.lib (with the linker command -lmylib).  The linker searches the ar/l
ib (libmylib.lib) to complete the linking.

The file created when the library is created (libmylib.lib) is called a static library.
  When a static library is linked the main program, it is fixed in the executable.  Thi
s has two consequences:

a) You can't change the library without relinking the whole program.
b) You don't have to be concerned about how the program finds the library at run-time (
no DLL hell).

Reasons for building a static library are:
  i) Code reuse across projects
     (one compilation of a library can be used by many other applications).
 ii) You don't need to distribute source code for the library
     (although you will have to distribute the library .h file)
iii) Library users require the library .h file to compile.

----

Linking type 3 is available on most operating systems.  Windows calls the dynamic libra
ry a DLL, Linus calls it a Shared Object (.so) and MacOS-X calls it a dylib.  Conceptua
lly they are identical.

This is very different from linking type 2.  In this form of linking, a stub is linked
with your main program.  At run-time, the system searches for the dynamic library and u
pdates the stub to call into the dynamic library.
```

This reverses a) and b) above as follows:

a) You CAN the library without relinking the whole program.
b) You have to be concerned about how the program finds the library at run-time (DLL he
ll).

Linking type 4 is MacOS-X centric. A Framework is a Bundle that contains a dynamic lib
rary. The Framework is a bundle which can hold other resources include graphics, prope
rty lists, other executables - almost anything really. A Framework is a directory in t
he File System.

I don't know that there anything very special about Framework linking. It amounts to t
he same thing as linking type 3. However it is both MacOS-X Centric and convenient. T
he Framework is a container for resources you wish to guarantee to be part of the Frame
work (for example the library header file).

Essential Commands to understand when working with libraries and build engineering task
s:

```
nm -g executable           dump the symbols of the executable
otool -L executable        dump the link table of an executable
install_name_tool          manage the install name table of an executable
lipo                       manage the architectures of an executable
xcodebuild                 process .xcodeproj files from the command line
```

The dynamic loader (man dyld) responds to various environment strings which will help y
ou debug problems with dynamic linking.

Once more, I don't have time at the moment to explain why you have to learn about this
stuff. I'll try to find time tomorrow to document my knowledge about these commands.

Scripts

I've included 3 scripts which I hope you'll find useful and interesting.

```
buildAll.sh Rebuilds the whole project
            It also lists the properties of the libraries (using ls, lipo and otool)
            It also run every variant of the console command-line tool.

Other scripts (for use by Robin).
manual.sh   Generates the documentation (console.pdf) for this project (from ReadMe.txt
)
            This depends on a2ps (I don't know if that's available on all Mac's)
publish.sh  Publishes this project on clanmills.com (useful only on my machine of cours
e)
```

Robin Mills
robin@clamills.com
http://clanmills.com
2010-09-19 17:42:01PDT