macoslib for Advanced Users

Document for **macoslib** v146.

You are *pretty* familiar with CFObjects (Carbon) and NSObjects (Cocoa) so using **macoslib** should be rather easy for you.

Installation in your Projects

1. Open the *macoslib.rbvcp* file with Xojo or Real Studio.
2. Select the "macoslib" folder inside the project.
3. Copy it by typing ⌘-C.
4. Create a new project.
5. Paste "macoslib" into it by typing ⌘-V.
6. Optionally:
   a. Go back to the *macoslib.rbvcp* project and copy the "Additional Modules" folder.[1]
   b. Paste it into your own project.

You are ready to go !

Note: you **cannot** drag&drop macoslib files from the Finder to the project window.

Using macoslib

Naming Conventions

For Carbon, the objects declared in macoslib are named by stripping off the *"Ref"* suffix of the Carbon object reference name, so CFArrayRef becomes CFArray.

For Cocoa, the objects are named after their normal name, so the Cocoa NSString class is just implemented as *NSString* in **macoslib**.

Hierarchy

macoslib folder:

BSD contains a very few BSD functions.
Cocoa contains all the NS… prefixed objects.
CoreFoundation contains all the CF… objects.
OS_Specific contains some OS specific code
Exception Classes contains exception definitions for the different OSes and for different exception origins (Mac OS, libc…)
Other modules contain either:

- Some Carbon modules declaring classes from different frameworks, e.g. CoreGraphics, CoreText.
- Other Cocoa objects not beginning with NS, e.g. QTKit or ImageKit objects.
- Old stuff

Additional Modules folder:

[1] The "Additional Modules" folder is based on the "macoslib" folder, so you need to paste it **after** "macoslib".

Contains optional convenience objects and modules which are based on macoslib but not necessary to macoslib.

Class Extensions: a collection of convenience extensions which take and return only Xojo native types.
Cocoa MenuItems: implementation of the common Cocoa MenuItems.
TT's SmartPreferences: handles app preferences. Works on OS X, Win and Linux.
KT's PList Stuff: Kem Tekinay's property list helper.
DebugReport Cross-Platform: a debugging system to follow execution of your code.

Implementation

We usually try to implement system objects in a way that is as close as possible to the native NSObject, while taking into account the limitations of Xojo. However, developers taking part to macoslib may also have different opinions about naming and implementation. The most obvious disagreements are:

1. In favor of or against exceptions.
2. Using *Constructors* or *Shared methods* to create a new instance (usually, both are implemented).

At the same time, some try to create some convenience methods so even newbies can use macoslib without any knowledge of Carbon or Cocoa. Such convenience methods should take and return only Xojo native types and be stored in the *Additional Modules* folder.

Inheritance

In Cocoa

Every NS class is based on NSObject except classes inheriting from NSControl, which is internally based on a Canvas.

NSObject implements 3 types of allocation, possibly extended by its children:

1. NSObject.Allocate(classPointer as Ptr) as Ptr
2. NSObject.Allocate(className as String) as Ptr
3. New NSObject(objectPointer as Ptr, hasOwnership as boolean = false) as Ptr

Also, NSObject automatically converts to Ptr by returning its *id* (the Cocoa *id* type uniquely determines an object).

Before you can create an instance of a class from a framework which is not loaded by default, call the *RequireFramework* method (you can call it several times) before instantiating the class.

Whenever possible, you should overload the *VariantValue() as Variant* method to return a sensible value in the Xojo environment. The default implementation returns *self*.

In Carbon

Every Carbon object is based on CFType. CoreFoundation objects which can be used in property lists also implement the CFPropertyList interface. There are: CFArray, CFBoolean, CFData, CFDate, CFDictionary, CFNumber, CFString and their mutable counterparts.

The main way of instantiation is Constructor(ref as Ptr, hasOwnership as boolean).

Mutable and Immutable Objects

In Carbon or in Cocoa, objects that hold data like strings, arrays, dictionaries, and so on cannot be modified, i.e. their contents cannot be modified; they are are said to be *immutable*, which is the default. Whenever you need to modify the contents of an object, you must get the *mutable* version of it, i.e. the "mutable" word must appear inside the name. For example, the mutable counterpart of NSArray is NSMutableArray (a subclass of NSArray).

You cannot make an immutable object become mutable. However, you can create a mutable *copy* of an immutable object by calling MutableCopy (inherited from NSObject).

If you Compile your Application for Carbon

Many functions in **macoslib** use Cocoa instead of Carbon. However, a Carbon application can use Cocoa seemlessly if your application executes the following code at startup in your *App.Open* event:

```
Cocoa.Initialize
```

How to Get more Information

1. Run the **macoslib** project and run the different examples from the *Examples* menu. It will be a good start to see how the Xojo objects handle their Carbon or Cocoa counterpart.
2. Have a look at the Apple's documentation and compare it to the **macoslib** objects. You should be able to easily understand how the API was implemented in **macoslib**.