macoslib Overview

macoslib is a collection of objects and functions filling the gap between Xojo and OS X. This document describes **only some** of the improvements macoslib brings to your apps.

Automatic multilingual interface: many system-wide items (helper windows, built-in menu items…) are automatically localized according to the System Settings in the *Language and Text* panel. In this document, any text which is not in English (usually it is in French), should be considered as automatically translated by the system. When too discrete, the stress can be put on some localizations with a colored ellipse.

The Purpose of macoslib

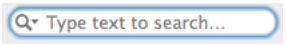
Xojo is mainly focused on being cross-platform, i.e. you can use the same code and generate an application for OS X, Windows or Linux. This is a very good idea but it also means that some features available on only one OS may be dismissed because they have no equivalent on the other OSes.

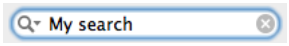
Whenever you want to use some "advanced" features of an OS, you need to use direct system calls through the *declare* statement. The problem is that not everyone has the skill to do that and that is where **macoslib** intervenes, at least for OS X. If you want your application to use the full power of OS X features then **macoslib** can be helpful.

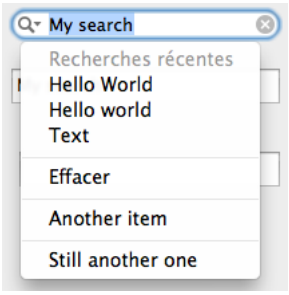
macoslib defines a lot of Carbon and Cocoa objects. Each object has methods and properties which will be directly sent to the system, so you are no longer limited by Xojo. Not only some Xojo classes are extended, but you can also use some classes/controls which are not implemented at all in Xojo.

Controls and Windows

Search Field





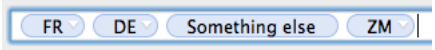


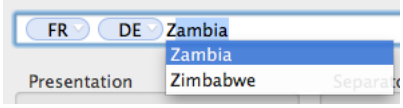
The famous Search Field, with its prompt, the cross icon to delete the text typed and the fully customizable menu.

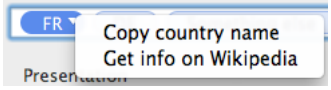
Recent searches can be easily included and there are automatically saved in the preferences file.

The Search Field class can also be inserted into Toolbars.

Token Field



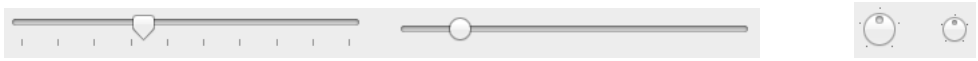




You have been using the Token Field for ages in *Mail.app* without knowing its name. Each value corresponds to a object but it is displayed as an abbreviation. When typing, an autocompletion menu is displayed (middle image). Also, each object can have its own menu (right image).

Objects can be rearranged by drag & drop and they natively support copying, pasting, deleting and undo.

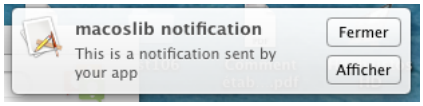
Sliders



Horizontal, vertical or round (knobs), sliders are everywhere. With **macoslib** you get all the flexibility OS X can provide.

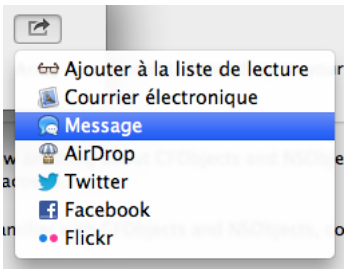
User Notifications

Since OS X 10.8, applications can display user notifications as Growl used to. Such notifications appear in the Notification Center but they can also be postponed, repeated…

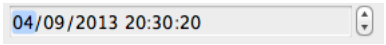


Share Button

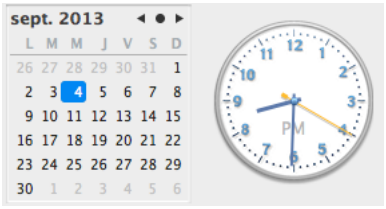
Inherited from iOS, the new Mountain Lion's Share button allows the end-user to share data through Facebook, Twitter, Flickr, AirDrop, an iMessage, an email or to add the item to the reading list of Safari. Just provide the Share button with the data to be shared and OS X will show a localized list of the possibilities and manage all the work (including the configuration, if necessary).



Date Picker



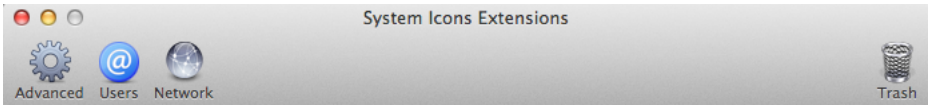
The Date Picker allows you to set a Date and/or a Time very simply and the way OS X does.



Accessing System Icons and Pictures

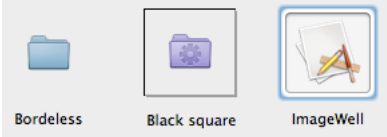
With a single line of code, you can access any system picture. It can be a Finder icon (folder, application...) or icons to be used inside buttons. As an example, the following buttons were created on-the-fly from system icons.

But you also can use those system icons in your Toolbar:



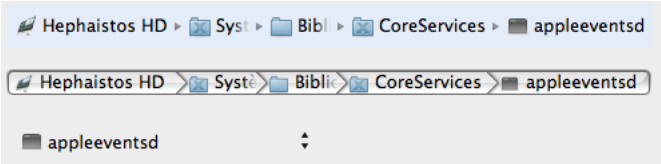
… and using the Image View control to display them

With the Image View control, you get rid off the hassle of painting, scaling or aligning the displayed picture. It has much more fine grained options than Xojo. Also, it can handle automatically the copy/paste/cut commands and the drag&drop operations.



Path Control

When working with the disk hierarchy, it may be a nice user experience to display an interactive path of a FolderItem. Your app is notified when the end-user clicks on one component of the path so you can update the content of your window.



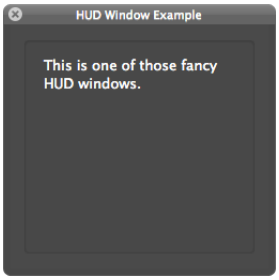
The three different styles:
- Standard
- NavigationBar
- PopUp

For folders whose name has been truncated, they automatically expand when the mouse passes over them.

DockTile Badge

Ever wanted to add a badge to your application icon in the Dock like Mail, App Store and many others ? **macoslib** provides such feature and it just takes 2 lines of code.

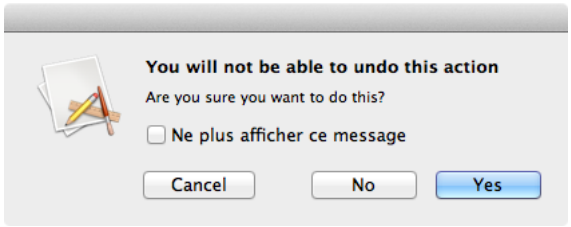
Windows Extensions



For most apps, windows are the heart of the interaction with the end-user.

The famous HUD (*Heads-Up Display*) is this floating dark gray translucent window everybody wants but it is not available even in the latest Xojo release (2013r2 at the time of writing). **macoslib** lets you create one very easily and it works exactly as any floating window.

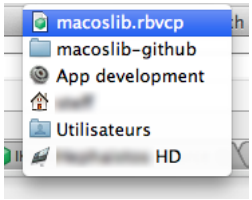
NSAlerts



Standard alert windows can now be created very easily and they are very flexible. You can even set the *"Do not show this message again"* checkbox (localized) whose value will be saved in your app preferences.

Proxy Icon and Menu

The absolute must-have for any document-based window is the so-called proxy icon, i.e. the document icon preceding the title. Not only does it enable the proxy menu to access the folder hierarchy to which the file belongs, but also it allows to drag&drop the file itself either to move/copy it or to create an alias to it. With **macoslib**, you can have all that with a single line of code !



When *right*-clicking or *Ctrl*-clicking on the title, the following menu should appear, listing all the localized folders in disk hierarchy up to the document file.

If you select an item, the corresponding folder will be displayed in the Finder.

Popover Windows

Popover windows can be used to display informative text and pictures (as Spotlight does), but they also can have all the controls you need. When using such windows to display a kind of toolbox, you may want to allow end-users to tear off the popover window which then becomes a regular window.

Full Screen Applications

OS X Lion introduced the concept of full-screen applications which are best used with several workspaces. Clicking on the icon on the right of the title bar brings you to that mode.

Translucent and Transparent Windows

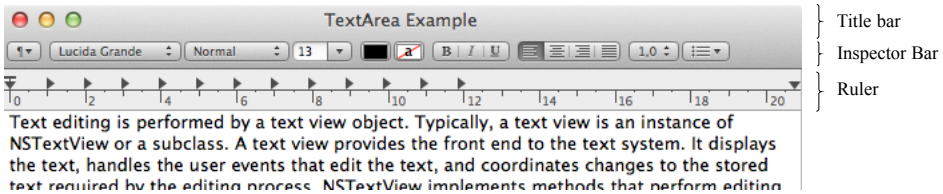
There are two ways to cope with windows transparency: either the whole window and its contents can become slightly transparent (like the Terminal windows), or only the window becomes more and more transparent while its contents remain opaque, which allows you to create windows of any shape like the **macoslib** splash window.

Remembering Windows Size and Placement

With **macoslib**, you can easily store and retrieve any window size and placement to/from the preferences file. Usually, it just takes two lines of code in the window's Open event.

Word Processing

You may think that a TextArea is just an area to type some plain text but it is so much more. Actually, a TextArea is what you can find in the TextEdit application. With the extensions provided by **macoslib**, you can display and use both the *"Inspector Bar"* and/or the *"Ruler"* (see below).



Other options include:

- Inserting lists with customizable prefix
- Inserting tables
- Inserting pictures (if the backend storage allows it)
- Controlling spelling and grammar correction
- Allowing user-defined automatic replacements
- Reading text through Speech
- Inserting links
- Using automatic link and/or data detection
- ... and many other things

Menus for Word Processing

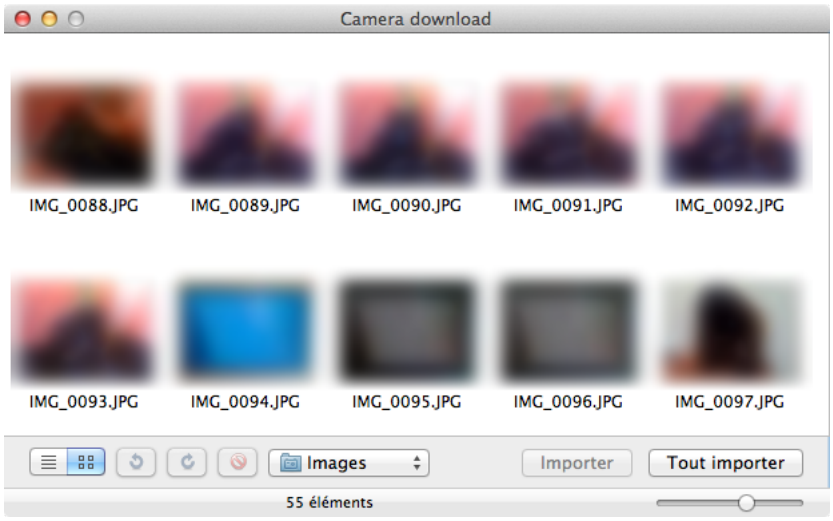
Along with the TextArea extensions, you can also install the standard Cocoa menus. Not only will they be automatically connected to TextAreas, but also they will be translated in the end-user's preferred language.

Managing Cameras and Scanners

Do you want to transfer photos from/to your camera/iPhone/iPad ? Or do you want to control your scanner (including multifunction printers) to preview and scan a document ? This is pretty easy with **macoslib**. You will get the standard display as in any Apple application (e.g. Preview), including the list of usable hardware locally and/or on the network. As a bonus, the interface is automatically localized per user's choice. See *ImageKit* and *ImageCaptureKit* in **macoslib**.

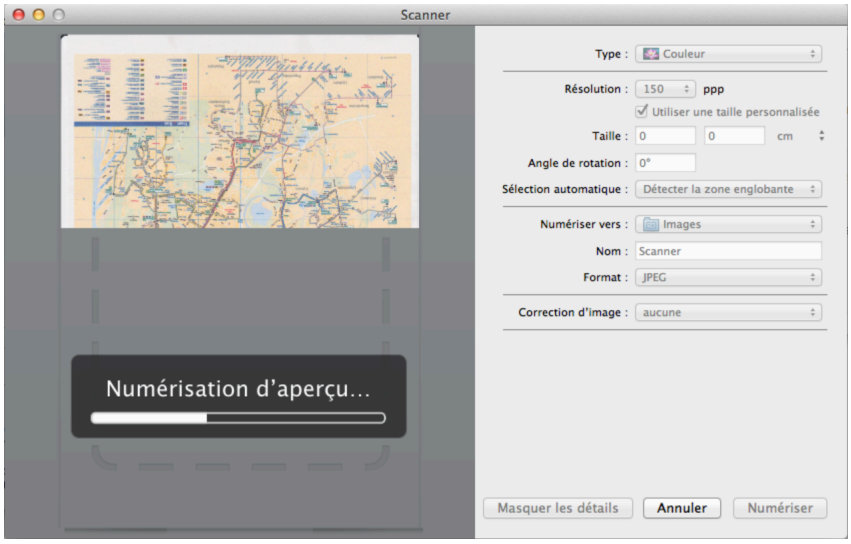
On the left, an example list of available devices (you can select a view by icons). *"NO NAME"* is an old camera with 40 items in it and it can be ejected right from your application, *"iPhone"* is obviously my iPhone with 55 photos and the *Canon MP240 Series* is a multifunction printer (here the scanner part is proposed).

For any camera/iDevice, either local or networked, you can access the contents of the device with several presentations (here a presentation by icons) and import one or more items. Also you can delete some items if your application and the device allows it.



This example shows a (blurred) part of my iPhone's content. The whole interface is automatically localized.

As a developer, you can choose to propose different features to the end-users: rotating pictures, deleting selected ones, choosing the destination folder for importation, setting the icons size…

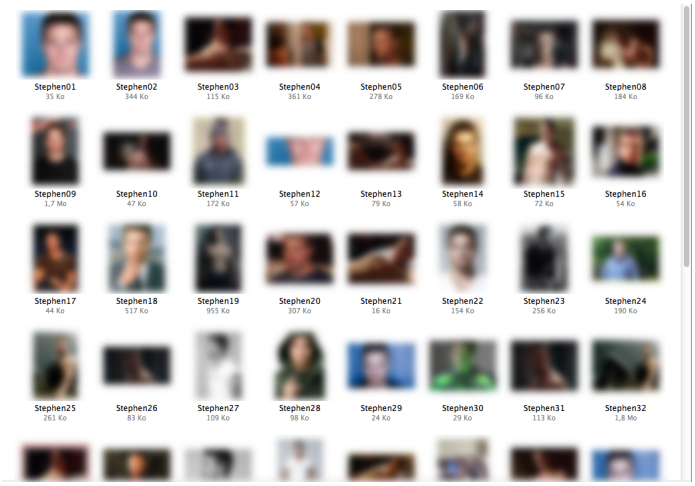


For a scanner or multifunction printer, you can access the standard scanner window (here, a snapshot of the window while it is acquiring the document preview). It is the same interface as the one you can see in the Preview application.

Note that the full interface is localized. You can also select which values can be changed by end-users.

Managing (a Large Amount of) Pictures

Just like you can get a window containing all the pictures stored on a camera/iPhone, it is possible to manage a large number of pictures with the same presentation with the *IKImageBrowser*. You just need to feed the IKImageBrowser with pictures whatever their origin.



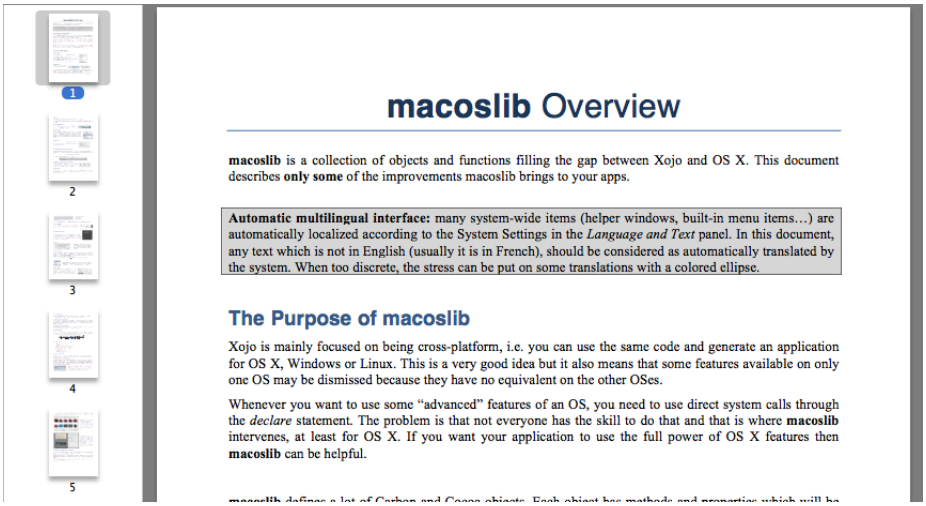
In the example on the right,[1] the name of each image item was set to the file's name without the extension and the subtitles (in gray) indicate the localized size of each file.

In the current implementation, you can invoke QuickLook by pressing the space bar, reorder as many items as you want (though drap&drop is not implemented yet).

[1] Images have been blurred due to copyright issues.

Managing PDF Documents

PDF management is at the core of OS X and several classes allows displaying and editing a PDF document. Below, an example of the current document (as PDF) displayed inside **macoslib**. On the right, the document is displayed inside a *PDFView*. On the left, a *PDFThumbnailView* (connected to the *PDFView*) automatically displays a thumbnail of each page and both controls work together.



Oh by the way, you would be surprized to see that such view can automatically **print** the PDF document !

What You can't See

Many classes and controls provide invisible features (but the best is always behind the scene !).

Doing Like the Finder

There are many cases where you would like to do things "like the Finder". Here are a few ways to achieve that goal:

Formatting Files Size

macoslib can format files size like the Finder. The result is 100% conform to Finder with OS X 10.8 and later but there is a compatibility layer to ensure that it also works decently on previous versions of OS X.

Sorting Filenames

You have noticed that the Finder sorts filenames by interpreting numerical values as numbers instead of strictly treating them alphabetically. Also, it can handle accented characters from different languages. **macoslib** has all the tools you need to propose the same behavior to your end-users.

Bonjour

If you are a Mac user, you must know Bonjour, a cross-platform technology to detect all the services available on the local network. For example, many printers implement it so they are automatically detected and configured by your Macintosh.

With **macoslib**, it becomes easy to detect any printer or service over the network, but also to advertise your own service.

Bonjour is also available on Windows (Bonjour™ for Windows), or on Linux through different packages (Avahi is the most common. You can find it at http://www.avahi.org if it is not already built-in your favorite Linux distro).

Files

There are several file properties that cannot be accessed through Xojo.

Access Control List (ACL)

The *Access Control List* (aka ACL) is a way to have a fine grained control over file/folder access, and it is prioritary over simple UNIX-like permissions. **macoslib** allows you to access/modify/delete ACLs.

Extended Attributes

Each file can have extended attributes, i.e. data fields attached to the file at the filesystem level. For example, OS X uses extended attributes to store the Resource Fork, Finder data or Quarantine data (i.e. all the stuff to warn you that a file has been downloaded from the internet ans ask you if you really want to open it).

But you can also add anything[2] as Extended Attribute, depending on what you need.

NSFileManager

The NSFileManager brings you access to low-level filesystem functions to get/set informations, scan folders contents up to 25 times faster than pure Xojo code, create symbolic and hard links…

File Icon

One line of code is all you need to get/set a custom icon for a file. It takes/returns a Picture with the corresponding mask so there is no limitations in the transparency.

Uniform Type Identifier (UTI)

The OS X way to identify a file type is to use a *Uniform Type Identifier*[3] (UTI) string. With **macoslib**, you can query such UTI for any FolderItem or determine if it inherits from a given UTI.

System Notifications

You may be interested in being informed whenever a system-wide event occurs like a new disk being mounted or ejected, the user having changed the Finder labels, an application being launched or terminated and so on. **macoslib** can do that for you.

Filesystem Events (FSEvents)

Each time a file or folder is created/modified/deleted, a new entry is created in the FSEvent (aka *FileSystem Event*) database. It allows you to know what is happening or what happened since the last time your application quit.

Speech

Speech synthesizing was quite limited before Lion but now, it is multilingual and supports high-quality voices. **macoslib** offers a fine-grained control over the voices, volume, rate and all the events associated with speech synthesis.

ResourceFork

While Xojo abandonned the ResourceFork object, it is still used in OS X. **macoslib** proposes a replacement for that.

Compilation Warnings

Contributors to **macoslib** always try to limit the number of warnings Xojo will display at compile time. Most of the time, such warnings correspond to *Unused method parameters, Unused local variables* or *Unused*

---

[2] The size of each Extended Attribute is limited to ca. 8000 bytes.

[3] For more information, see http://docs.xojo.com/index.php/UTI .

event parameters. They are avoided on every platform by adding *#pragma unused* statements wherever necessary. As such, compilation of **macoslib** should raise almost no warnings.

Additional Modules

DebugReport (by SM)

DebugReport is an easy way to display a list of values of any kind (any value from Xojo, Carbon or Cocoa) either in real time[4] or until its window is refreshed. You can display a title, a warning or an error, each being displayed with a different color. You can also scroll to the next/previous warning or error with a single click. Whenever your project raises an exception and falls into the debugger, you can still see the contents of the debug log as it is stored internally by going to Globals→DebugReportModule→LogText. Although you may prefer that each debugging message is echoed to the system so you can analyze it later. It is up to you.

While other logging solutions often ask you to specify the type of value you want to log, either by using only strings or by using separate methods to log an integer, a double or a string, **DebugReport** automatically formats **any** values you pass to it or **array** of values. You can also create a custom formatter for your own classes.

You do not even need to remove your debugging code when building your application as **DebugReport** deactivates itself on building the application, i.e. it is not even compiled. But if you need it nonetheless, adjusting one or two constants will be enough. You will never need to change the code.

DebugReport can now also report NSNotifications on Cocoa. Simply register the notification name, the target object or both and the logging window will do the rest for you with all the details needed.

PropertyList

DebugReport contains its own cross-platform implementation of XML PropertyLists which can also be used from outside DebugReport.

[4] Note that, compared to a normal execution, forcing the debug window to refresh all the time very seriously impairs the performances, especially in tight loops. Pressing both Ctrl-Option will block force-refreshing whenever needed.