Quick Start
from March 13, 2012

This first simple 'Objective-Basic' program will familiarize you with the language, as well as with Cocoa and the Integrated Development Environment (IDE). Follow the steps and read the explanations below and you will find a whole new world to explore!

We estimate that you will need about an hour to complete this Quick Start Manual.

First Things First

Let's go

Start 'Objective-Basic' by double clicking on the symbol on your desktop.

Create a new project

On the menubar, click *File* and then *New Project* (from now on, menu selections will be listed like this: *File* | *New Project*). Enter a short name for your project, such as **Quick Start** (from now on, anything you type from the keyboard will be **bolded**). Whatever you decide, a new directory will be created with that name, into which all project files will be saved.

Don't use '/', '.', or other special characters. For now, just use numbers and letters, and you may add spaces between words.

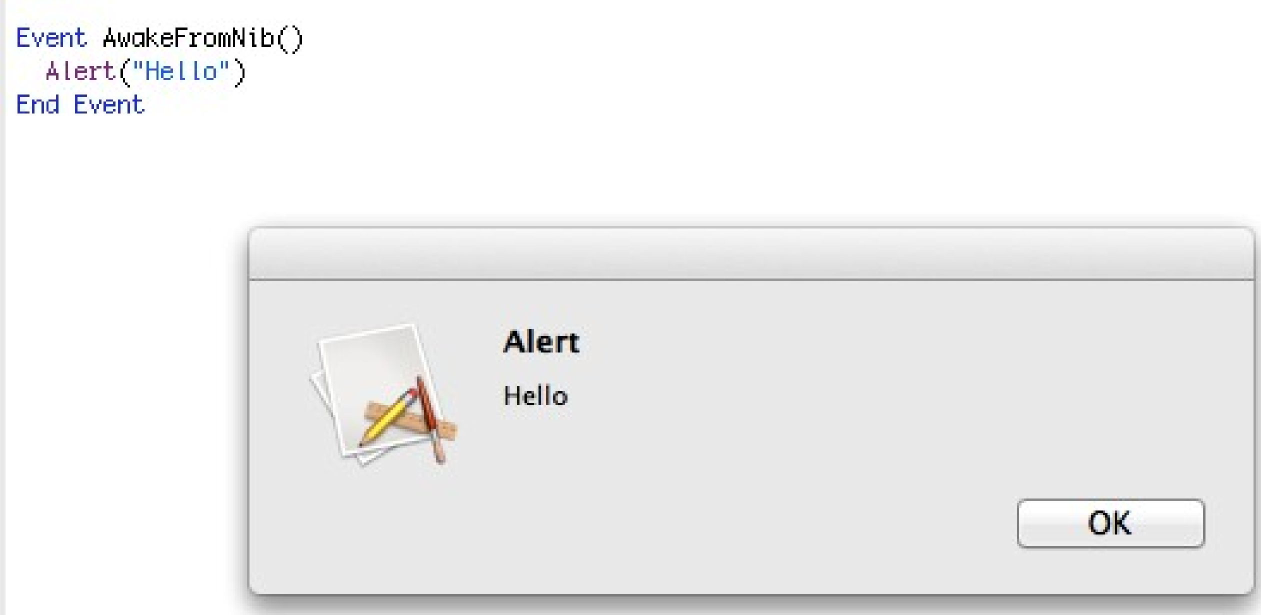
After clicking *OK*, a new project will be created for you, normally on the `Desktop` of the current user (from now on, all folder and file names, as well as text on the computer monitor will be in the `Courier` font).

Using 'Finder', you can look at the folder and you will see just one file, `Global.NSObject.objb` (normally just called 'Global'). This is a special file automatically created by 'Objective Basic' with the a few instructions – called a procedure – beginning with `Event AwakeFromNib().`

The `Global.NSObject.objb` file is the file in which you may declare variables, constants, and functions to be used as global elements of your program. If you understand something of object-oriented programming, it will interest you to know that these elements do not need to be created as objects beforehand, similar to a module in 'Virtual Basic'. The `Event AwakeFromNib()` procedure is called after 'Cocoa' automatically loads `MainMenu.xib` on startup of your application. You may freely add code to it, but we won't do much with it in this small tutorial.

The first run of your application

Let's run this small example and see what happens. Either click the run button in the toolbar of the Objective-Basic IDE (▶), or in the menu bar, clicking on *Compiler* | *Run*.



```
Event AwakeFromNib()
  Alert("Hello")
End Event
```

Alert

Hello

Screenshot 1: The screen of your Mac will look like this after running your new program.

You probably noticed the command `Alert("Hello”)` in the `Event AwakeFromNib()` procedure. This command makes the Alert window appear with the text `Hello` when you run your application.

Stopping your application

When you click the *Stop* button in the toolbar (✕), select *Compiler* | *Stop* in the menu bar*,* your application will immediately stop execution.

Editing your application

Let us change the `AwakeFromNib` procedure to "Hello World!”. Delete the line `Alert("Hello”)` and type in **Alert("Hello World!”)**. Run the example again as instructed “The first run of your application” section above. Now `Hello World!` appears instead of `Hello`.

You have now edited your first 'Objective-Basic' application. Congratulations!

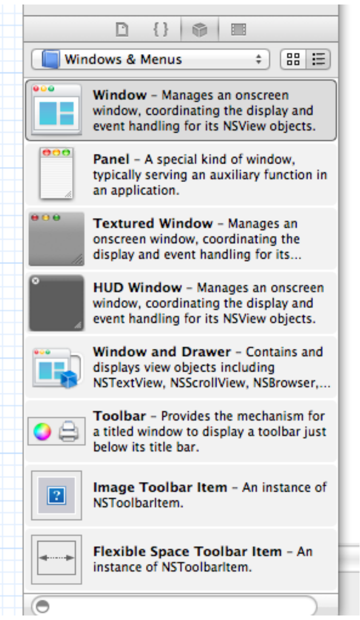
Working With Interface Builder (IB)

Windows and buttons

Now it's time to try a more sophisticated way to put a window on the screen. You will start 'Interface Builder' (IB) and use it to create a new window with a button on it. We will program the application to change the button's title and to show a message, when the button is clicked. To get a quick overview about 'IB', read the documentation provided with 'Objective-Basic' entitled 'Xcode hints'. Click on *Help* | *Xcode hints* in the menu bar of the 'Objective-Basic' IDE.

Please note that the following instructions have been verified under 'Xcode®' 4.3, in which 'IB' is *integral to the 'Xcode®' IDE. In 'Xcode' version prior to 4.x, 'IB' was closely linked to 'Xcode', but still a separate application. This may create slight differences in the following instructions.*

Xcode®/Interface Builder (X/IB)



Windows & Menus

Window – Manages an onscreen window, coordinating the display and event handling for its NSView objects.

Panel – A special kind of window, typically serving an auxiliary function in an application.

Textured Window – Manages an onscreen window, coordinating the display and event handling for its...

HUD Window – Manages an onscreen window, coordinating the display and event handling for its NSView objects.

Window and Drawer – Contains and displays view objects including NSTextView, NSScrollView, NSBrowser,...

Toolbar – Provides the mechanism for a titled window to display a toolbar just below its title bar.

Image Toolbar Item – An instance of NSToolbarItem.

Flexible Space Toolbar Item – An instance of NSToolbarItem.

Screenshot 2: Object Library

To start 'IB' , select *Project* | *Xcode for MainMenu.xib* in the menu bar. 'IB' will load with the default Graphics User Interface (GUI) file `MainMenu.xib`. You are now in 'X/IB'.

In order to create a new window, do this.

1. In the menubar of 'Xcode', click *View* | *Utilities* | *Show Object Library*.
2. Search the list for the element we need by clicking on *(Object) Library* | *Cocoa* | *Windows & Menus* in the `Object Library`.
3. An icon list will appear beneath the `Windows & Menus` selection you just made. The first item in the list is `Window`. Double-click `Window` and a new window will be inserted in the project list of `MainMenu.xib`.

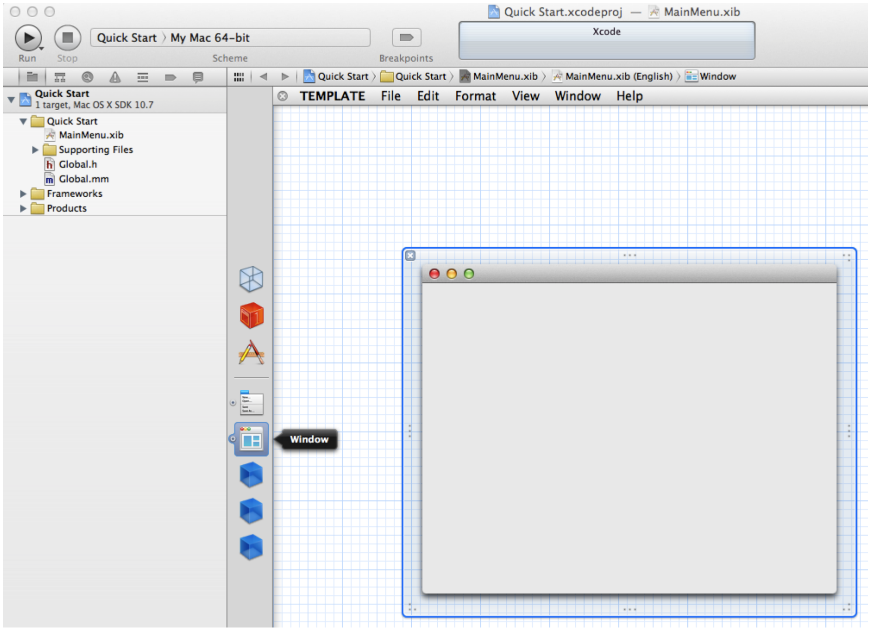
New Window

You will normally find that an empty window is already available in your `MainMenu.xib`. But if you like, you can create another one. Find the project file list named `MainMenu.xib` and double-click on the newly created window shown as `Window`. The window will appear as empty window on screen, ready to be extended by dropping controls on it (see the screenshot on this page).

New Button

Place a button on the empty window. Again using the Object Library window, select *(Object) Library* | *Cocoa* | *Controls* | *Buttons*. In the icon list below, select `Push Button`, dragging and dropping the button onto newly created window.

You can then click on the border of the button and resize it. Or you can click in the middle of the button and drag it to a new location.


Quick Start.xcodeproj — MainMenu.xib
Run
Stop
Quick Start › My Mac 64-bit
Scheme
Breakpoints
Xcode
Quick Start › Quick Start › MainMenu.xib › MainMenu.xib (English) › Window
Quick Start
1 target, Mac OS X SDK 10.7
Quick Start
MainMenu.xib
Supporting Files
Global.h
Global.mm
Frameworks
Products
TEMPLATE File Edit Format View Window Help
Window

Basic usage of Xcode's Interface Builder (X/IB)

With 'IB', once you have drawn your windows and controls, you can connect them to variables and functions that you will program into your source code in the 'Objective-Basic' IDE.

Next Step

First, save the file opened in 'Xcode's' 'Interface Builder'. In the menu bar of 'Xcode', select *File | Save.*

Now that you have created a window and put a button it, you need to tell 'Objective-Basic' what you want those controls to do. But before we take that next step, let's run your example again.

Switch to the 'Objective-Basic' IDE and then select *Compiler | Run* from the IDE menu bar. Several things will happen.

1. The Alert box with your message will appear again.
2. But after clicking `OK`, your newly created window with the button will appear.

You just managed to get a custom Cocoa-based window to appear on your screen!

Connecting Code And Objects

Writing code for the window

Code is always written in the IDE of 'Objective-Basic'. We now need to write code for the window that was created in 'Xcode' by creating a new source code file in the 'Objective-Basic' IDE. Do this:

1. Using the menu bar, select `Project | New File With Super Class | NSObject`.
2. Enter **code** as the name.
3. Add this new line in the empty window: **IBOutlet mybutton As NSButton**. It doesn't matter what line you place the code on. The code declares a variable called `mybutton`. The rest of the source code in the project will use the variable to control the button object you created in the `MainMenu.xib` file.
4. Add these following lines.

```
IBAction myevent(sender As id)
        mybutton.Title = "Just changed the title"
End IBAction
```

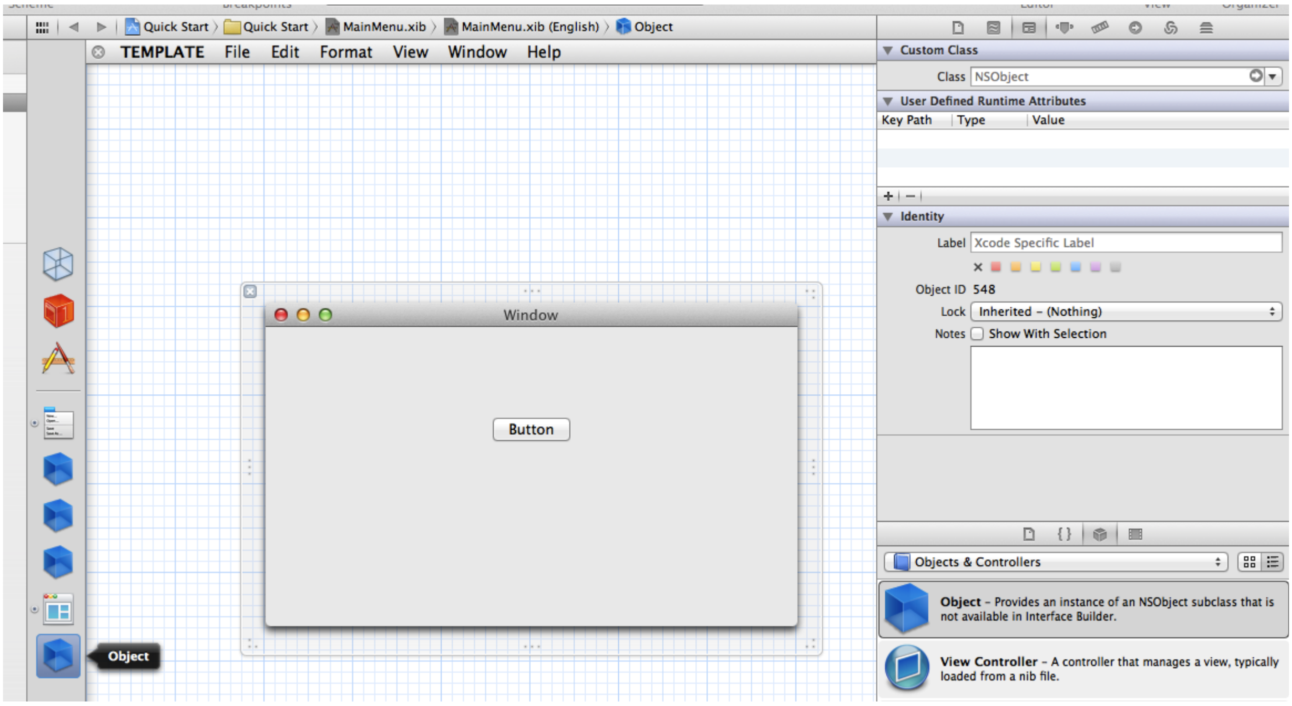
These lines create an event-triggered procedure used for the 'Interface Builder' objects you have created. “Event-triggered” means that when the button is clicked (the “event”), the above code is triggered (or “called”).

Finally, click the `Build` button (↻) (or use the menu bar to select *Compiler | Build*) to inform 'IB' about the changes in your source code. Do this whenever you add or remove 'IB'-related code in the 'Objective-Basic' IDE. The current version of the 'Objective-Basic' IDE automatically updates the changes in X/IB, when you use `IB` button on the menu bar, or switch to X/IB by selecting *Project | Xcode ...* on the 'Objective-Basic' menu bar.

Connecting The Object

Writing those lines was part one. Now you need to tell 'X/IB' about the source code you have written. We need to make use of the code file `code.NSObject.objb` in 'IB' by telling IB to use the file when the `MainMenu.xib` file is loaded. `code.NSObject.objb` is actually a class, from which usable objects are created (“instantiated” in object-oriented parlance). To do that, we need to create a new empty object and add to the list of objects.

1. First, switch to X/IB.
2. Use the Xcode® menu bar and select *View* | *Utilities* | *Show Object Library*.
3. Select *Library* | *Cocoa* | *Objects & Controllers* in the `Object Library`.
4. From the icon list beneath `Objects & Controllers`, select `Object` by double clicking on it. A new object will be inserted in the project list of `MainMenu.xib`.
5. Select this newly created object and change the class of that object by typing **code** in the `Custom Class` box. It is not important that you know what exactly class means here. We may see it as the file name used for the creation of objects. In our case, the file name is `code.NSObject.objb`, the “class name” we created in 'Objective-Basic'.



Class NSObject

Label Xcode Specific Label

Object ID 548

Lock Inherited – (Nothing)

Notes ☐ Show With Selection

Object – Provides an instance of an NSObject subclass that is not available in Interface Builder.

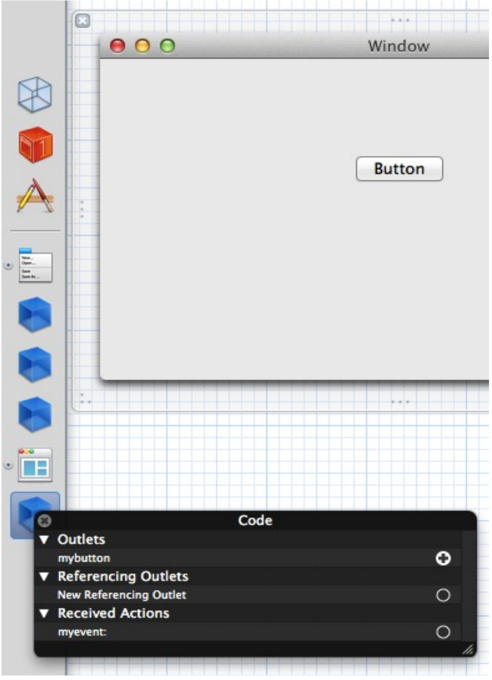
View Controller – A controller that manages a view, typically loaded from a nib file.

Screenshot 3: The property window is used to change the class name of the object. Select Identity Inspector *in the property window (upper right) to see the class identity.*

Telling 'X/IB' about your coded IBOutlet and IBAction

We are nearly finished with your example. The last task is to connect the `IBOutlet` and `IBAction` procedures you wrote in the source code file in 'Objective-Basic', with the button created in the window in 'X/IB'. Do this:

1. For `IBOutlet`:
    - (a) Open the context menu for the newly created object `code` in the `MainMenu.xib` window by left-clicking on it.
    - (b) If necessary, click on the triangle-shaped arrow next to Outlets. You will see the reference to the `mybutton` variable you created.



   (c) Draw a line from the circle to the right of `mybutton` to the button on the window you created in 'IB'.

2. For `IBAction`: The idea is exactly the same. Draw a line from `myevent:` in the context menu of the `code` object to the button.

You can think of `IBOutlet` as an "outlet" from your code in 'Objective-Basic' to a GUI object in 'X/IB'. `IBAction` is the "action" that will be performed, when the GUI object in 'X/IB' is triggered – in this case, the button.

Now, let's text your program!

1. Save the project in 'Xcode'.
2. Switch to the 'Objective-Basic' IDE and run your example.
3. Here's what should happen.
   (a) The `Alert` box will appear with your "Hello world!" message.
   (b) When you click the `OK` button in the Alert box, the box will disappear and the window you created will appear.
   (c) In that window, your button will read `Button`.
   (d) However, when you click your button, the name of the button will change to "Just changed the title," assuming you followed the instructions above.

Closing Final Tip

You can use your ESC key to get the list of all the possible commands with code completion.

To be continued and ... have a lot of fun!

Copyright

Copyright © 2007 - 2012 by www.objective-basic.com.

Products named herein are trademarks of their respective owners.