First Steps in GNUstep GUI Programming: NSApplication, NSMenu

Nicola Pero n.pero@mi.flashnet.it

July 2000 AD

This tutorial will introduce you to using `NSApplication` and `NSMenu`. *Warning:* You need GNUstep core libraries version 0.6.6 or later to try out the examples.

1 The shared application object

1.1 Creating the shared NSApplication instance

The class `NSApplication`, provided by the GNUstep GUI library, represents a gui application. Each gui application has one (and only one) instance of this class, which is shared by all the code; it keeps tracks of all the application windows and panels, and manages the application's run loop. You access this shared instance by calling the `+sharedApplication` method of `NSApplication`, which creates the instance of `NSApplication` representing your app the first time it is invoked, and returns the previously created instance when called again. Creating the shared application is very important, because when it is first created the gui library initializes the gnustep backend; in other words, you need to create the shared application object before doing anything at all with the gui or backend (xgps/xdps) library. So, we will start our first gui application with the code:

```
NSApplication *myApplication;

myApplication = [NSApplication sharedApplication];
```

An interesting thing to know is that, *after* you have created an `NSApplication` shared instance for your app, you can access it simply through the global variable `NSApp`. So, many people simply discard the result of `+sharedApplication`, and start their apps as follows:

```
/* The following line creates the shared application instance */
[NSApplication sharedApplication];

/* Then, use NSApp to access NSApplication's shared instance  */
```

1.2 Setting a delegate for the shared NSApplication instance

Once you have learnt how to create your application's shared instance, the fastest way to develop an application is to set a *delegate* for your shared application object. This is done by using the method `-setDelegate:`, as follows:

```
id myObject;

// <missing: create myObject>
[NSApp setDelegate: myObject];
```

A delegate is an object of your choice which can customize the behaviour of your application by implementing some (predefined) methods. For example, if you want your application to display a menu (all apps should), you just need to implement in the delegate a method called

```
- (void) applicationWillFinishLaunching: (NSNotification *)not;
```

(ignore the argument for now) and put the code to create the menu inside this method. Your shared application will check if the delegate of your choice has implemented this method, and if so it will run the method just before entering the main run loop. The documentation of `NSApplication` lists all the other methods the delegate can implement to customize your application's behaviour; we'll learn about some of them in other tutorials.

After you set the delegate of your application object, you need to run your application; to do this, just invoke the function `NSApplicationMain ()`, which does all for you.

To sum up, here is the code we are going to use:

```
#include <Foundation/Foundation.h>
#include <AppKit/AppKit.h>

@interface MyDelegate : NSObject
- (void) applicationWillFinishLaunching: (NSNotification *)not;
@end

@implementation MyDelegate
- (void) applicationWillFinishLaunching: (NSNotification *)not
{
  // TODO - Create the menu here.
}
@end

int main (int argc, const char **argv)
{
  [NSApplication sharedApplication];
  [NSApp setDelegate: [MyDelegate new]];

  return NSApplicationMain (argc, argv);
}
```

1.3 The run loop of your application

When you call `NSApplicationMain ()`, the GUI library "runs" your application. When the application is running, the GUI library simply enters a loop waiting for events from the user (such as the user clicking on a window, on a menu, or on the application icon). It creates an autorelease pool at the beginning of each loop, and empties it at the end; this means you don't have to worry about manually creating and emptying autorelease pools in your gui application once it is running.

Of course, if you run an application without having created any windows or menus, the application will wait indefinitely since the user has no means of communicating with it.

1.4 Terminating your application

Whenever you want to terminate your application, you can do:

```
[NSApp terminate: nil];
```

This is usually done when the user selects the `Quit` entry in the main menu. When you terminate your application, the gui library quits the run loop and exits the program.

Btw, the argument of `terminate:` is of no importance, so we can pass any argument (`nil` in this case); the only reason why this method takes an object as argument is that it can then be set as the `action` of a gui control (discussed below).

2 Setting your application's icon

To set an icon for your app, you just need to define in your `GNUmakefile` the variable `xxx_APPLICATION_ICON`, where `xxx` is the name of your application. For example, if we put the source code for our application shown before (still to be completed though) in the file `MyApp.m`, and the application icon in `MyApp.png`, the `GNUmakefile` could be as follows:

```
include $(GNUSTEP_MAKEFILES)/common.make

APP_NAME = MyApplication
MyApplication_APPLICATION_ICON = MyApp.png
MyApplication_RESOURCE_FILES = MyApp.png
MyApplication_OBJC_FILES = MyApp.m

include $(GNUSTEP_MAKEFILES)/application.make
```

Please note that you need to list the image in the resource files because you want it to be installed in the application directory (otherwise it can't be found by the app at run-time).

3 Menus

We are now going to add a menu to our tutorial application.

3.1 Creating a Menu

An `NSMenu` object represents a menu. You create a new menu simply with:

```
NSMenu *menu;

menu = AUTORELEASE ([NSMenu new]);
```

This is an empty menu. To add items to it, you do as follows:

```
[menu addItemWithTitle: @"Quit"
      action: @selector (terminate:)
      keyEquivalent: @"q"];
```

This adds to the `menu` menu an entry with title `Quit` (the title is the string which is displayed to the user). It's interesting to know that the method

```
addItemWithTitle:action:keyEquivalent:
```

actually returns a `NSMenuItem` object, which is the menu item object which was added to the menu. We don't need this return value in this example, so we discard it. Later on, we will need it.

3.1.1 Action and Target

A menu item behaves as most controls in the GNUstep gui library: it has an *action* and a *target*. The typical example of such a control is a button. Say, for example, that you have created a button called `myButton`. You want something to happen when the user clicks on the button - to do it, you need to specify an *action* and a *target* for the button. For example, if you want the method `terminate:` of `NSApp` (the application shared object) to be invoked when the user clicks the button, you do as follows:

```
NSButton *myButton;
id myObject;

// <missing code: create myButton, myObject etc>

[myButton setAction: @selector (terminate:)];
[myButton setTarget: NSApp];
```

When the user presses `myButton`, the gui library sends the `terminate:` message to the object `NSApp`, passing as argument `myButton`, executing the equivalent of the following code:

```
[NSApp terminate: myButton];
```

This way, when the user clicks the button, the application quits.

The button which was clicked is passed as an argument to the *action* so that the receiving object can determine (if needed) which button was actually pressed, so that the same method may be used for more than one button. In our case, `NSApp` simply discards the argument of `terminate:`.

In brief, the *action* is the method to be invoked when the user clicks on the button; the *target* is the object on which to invoke the method. *action* should be a selector for a method returning `void` and taking a single argument (of type `id`, a generic object). The button which was clicked is passed as an argument to the invocation of *action* on the *target*.

The case of a menu item is completely similar. When the user selects this menu item, the gui library performs the menu item *action* (`terminate:` in this case) on the menu item *target*. The argument passed to `terminate:` is the menu item which was selected.

Usually, no target is specified for a menu item, as in our code

```
[menu addItemWithTitle: @"Quit"
      action: @selector (terminate:)
      keyEquivalent: @"q"];
```

which only specifies that the action is `terminate:`. When no target is specified, the gui library tries to determine an appropriate target dynamically at run-time. For now the only important thing to know is that the gui library will try to send the action to certain objects (roughly, the objects inside a window which have the input focus) and, failing these objects, it will try to send the action to `NSApp`, and as a last resort to `NSApp`'s delegate. In our example, we want the action `terminate:` to be sent to `NSApp` (thus terminating the application when the user selects the `Quit` menu item), so this automatic mechanism works just fine for us (actually, you will soon discover that this automatic mechanism works fine extremely often). So, we don't need to set explicitly a target for our menu item.

3.2 Setting the Application Menu

Once you have created an `NSMenu` called, say, `myMenu`, to set it as the application menu you just invoke:

```
[NSApp setMainMenu: myMenu];
```

3.3 Our First Application

We are ready now to create our first gui application:

```
#include <Foundation/Foundation.h>
#include <AppKit/AppKit.h>

@interface MyDelegate : NSObject
- (void) applicationWillFinishLaunching: (NSNotification *)not;
@end

@implementation MyDelegate : NSObject
- (void) applicationWillFinishLaunching: (NSNotification *)not
{
  NSMenu *menu;

  menu = AUTORELEASE ([NSMenu new]);
  [menu addItemWithTitle: @"Quit"
        action: @selector (terminate:)
```

```
        keyEquivalent: @"q"];
  [NSApp setMainMenu: menu];
}
@end

int main (int argc, const char **argv)
{
  [NSApplication sharedApplication];
  [NSApp setDelegate: [MyDelegate new]];

  return NSApplicationMain (argc, argv);
}
```

Put this code for example into a file called `MyApp.m`, and use the following `GNUmakefile` for it:

```
include $(GNUSTEP_MAKEFILES)/common.make

APP_NAME = MyFirstApp
MyFirstApp_OBJC_FILES = MyApp.m

# Uncomment the following if you have an icon
#MyFirstApp_APPLICATION_ICON = MyApp.png
#MyFirstApp_RESOURCE_FILES = MyApp.png

include $(GNUSTEP_MAKEFILES)/application.make
```

Compile it, then run it using `openapp MyFirstApplication.app` (see the GNUmakefile mini-tutorial for further information on writing GNUmakefiles).

3.4 Fun with NSMenuItem's target

The next step in our tutorial is to add to the menu an item which prints `Hello!` to the user when the user selects the item.

So, we add a new menu item to our menu, invoking the `printHello:` action (which we'll implement in our custom object):

```
[menu addItemWithTitle: @"Print Hello"
      action: @selector (printHello:)
      keyEquivalent: @"h"];
```

Since our custom object is the application's delegate, we don't need to set explicitly the target: the library can determine it at run-time. In other cases it could be necessary to set a different target, as in:

```
NSMenuItem *menuItem;
id myObject;

// <missing code: create myObject etc>

menuItem = [menu addItemWithTitle: @"Print Hello"
                 action: @selector (printHello:)
```

```
                 keyEquivalent: @"h"];
[menuItem setTarget: myObject];
```

But in this case, we don't need to set the target explicitly, and the code is simply:

```
#include <Foundation/Foundation.h>
#include <AppKit/AppKit.h>

@interface MyDelegate : NSObject
- (void) printHello: (id)sender;
- (void) applicationWillFinishLaunching: (NSNotification *)not;
@end

@implementation MyDelegate : NSObject
- (void) printHello: (id)sender
{
  printf ("Hello!\n");
}

- (void) applicationWillFinishLaunching: (NSNotification *)not
{
  NSMenu *menu;

  menu = AUTORELEASE ([NSMenu new]);

  [menu addItemWithTitle: @"Print Hello"
        action: @selector (printHello:)
        keyEquivalent: @""];

  [menu addItemWithTitle: @"Quit"
        action: @selector (terminate:)
        keyEquivalent: @"q"];

  [NSApp setMainMenu: menu];
}
@end

int main (int argc, const char **argv)
{
  [NSApplication sharedApplication];
  [NSApp setDelegate: [MyDelegate new]];

  return NSApplicationMain (argc, argv);
}
```

The `GNUmakefile` is the same. I hope you appreciate how easy and simple is coding in GNUstep; I encourage you to try it out and enjoy all the fun you of selecting the `Print Hello` menu item and see the program print out `Hello!`.

3.5 Creating sub-menus

Creating sub-menus is quite easy. For example, to add an `Info...` sub-menu to your main application menu, you first create a menu representing it:

```
NSMenu *infoMenu;

infoMenu = AUTORELEASE ([NSMenu new]);
[infoMenu addItemWithTitle: @"Info Panel..."
          action: @selector (orderFrontStandardInfoPanel:)
          keyEquivalent: @""];
[infoMenu addItemWithTitle: @"Help..."
          action: @selector (orderFrontHelpPanel:)
          keyEquivalent: @"?"];
```

Then, you create an item in the main menu for your info menu, but instead of setting an action and a target for that item, you set the `infoMenu` as the sub-menu corresponding to that item:

```
NSMenuItem *menuItem;

menuItem = [menu addItemWithTitle: @"Info..."
                 action: NULL
                 keyEquivalent: @""];
[menu setSubmenu: infoMenu  forItem: menuItem];
```

3.6 Creating a standard info panel

You may have noticed that in our previous example with the `Info...` sub-menu, we have used `orderFrontStandardInfoPanel:` as the action for the `Info` menu entry. The `Info` menu entry is usually supposed to display an “Info Panel” (also called “About Panel” on Microsoft Windows), with the title of the program, the version, the author, the copyright info. In this case, we use `orderFrontStandardInfoPanel:`, which is implemented by `NSApplication` (it is a GNUstep extension), and which displays a standard info panel. The information on what to display in the panel is taken from the `Info-gnustep.plist` file in the application's main bundle. This file is created automatically for you at compile time by the GNUstep make system; but you can insert your own entries in this file as follows.

If your application name is, for example, `MyFirstApp`, then you need to create a file called `MyFirstAppInfo.plist` in your source directory (you do not need to add anything to your `GNUmakefile`; the make system looks for this file automatically). Here is an example of such a file:

```
{
  ApplicationName = "My First Application";
  ApplicationDescription = "An Example of how to use NSMenu";
  ApplicationRelease = "0.1";
  Authors = ("Nicola Pero <n.pero@mi.flashnet.it>",
             "John <john@john.it>");
  Copyright =
    "Copyright (c) 2000 Nicola Pero <n.pero@mi.flashnet.it>";
```

```
  CopyrightDescription =
    "This program is released under the GNU GPL";
}
```

You should of course edit this example filling in with the information appropriate for your own app. The file is in a format called “property list” (that is why it has extension `plist`; see the *Basic Foundation Classes* GNUstep Mini Tutorial for more information on property lists). The entries should be self-explanatory; note that `Authors` should be equal to an array of names. If you want to omit some of the entries, you may safely do it.

Here is the full listing of our latest app source code, containing an info submenu able to display the Info Panel:

```
#include <Foundation/Foundation.h>
#include <AppKit/AppKit.h>

@interface MyDelegate : NSObject
- (void) printHello: (id)sender;
- (void) applicationWillFinishLaunching: (NSNotification *)not;
@end

@implementation MyDelegate : NSObject
- (void) printHello: (id)sender
{
  printf ("Hello!\n");
}

- (void) applicationWillFinishLaunching: (NSNotification *)not
{
  NSMenu *menu;
  NSMenu *infoMenu;
  NSMenuItem *menuItem;

  menu = AUTORELEASE ([NSMenu new]);

  infoMenu = AUTORELEASE ([NSMenu new]);

  [infoMenu addItemWithTitle: @"Info Panel..."
            action: @selector (orderFrontStandardInfoPanel:)
            keyEquivalent: @""];

  [infoMenu addItemWithTitle: @"Help..."
            action: @selector (orderFrontHelpPanel:)
            keyEquivalent: @"?"];

  menuItem = [menu addItemWithTitle: @"Info..."
                   action: NULL
                   keyEquivalent: @""];

  [menu setSubmenu: infoMenu  forItem: menuItem];
```

```
  [menu addItemWithTitle: @"Print Hello"
        action: @selector (printHello:)
        keyEquivalent: @""];

  [menu addItemWithTitle: @"Quit"
        action: @selector (terminate:)
        keyEquivalent: @"q"];

  [NSApp setMainMenu: menu];
}
@end

int main (int argc, const char **argv)
{
  [NSApplication sharedApplication];
  [NSApp setDelegate: [MyDelegate new]];

  return NSApplicationMain (argc, argv);
}
```