Small App - Menus

In this chapter, we get started on our "small" MacOS X application framework. The goals of this framework design effort will be laid out, some code written, and some application and menu classes created. We will finish the chapter with a look at what we will need next.

Small App

The Small App project is an effort to design a minimal set of classes that will allow MacOS X programming novices to learn how to program for a modern Macintosh computer. These classes will represent elements of a modern program such as windows, menus, and the application itself. Such a set of classes is called an application framework and the goals of this framework are:

1. SMALL! - Too many times an application framework tries to be simple yet powerful and flexible. These are often mutually exclusive goals. The Small App framework is intended to be the minimal set of classes that will let a novice construct a standard GUI application with menus and windows with buttons, text, and images. The set of classes, their attributes, and their methods should be few enough in number that a student could actually learn them all.

2. Modern - This framework should incorporate all of the latest (at least MacOS X 10.3) system routines and their intended design. This means this framework will use the Carbon Event model, CoreFoundation, the HIView APIs, and CoreGraphics. It will also be "localizable" which means that resources will be added to the project to allow the application to use the local language and character set of its user. The resulting application will be linked against the Carbon framework and so it will be a "Carbon" application. Cocoa applications will be covered in a different Marten framework.

3. Editor Capable - The Marten IDE allows a user to construct object editors for their code from their code base. Typically, the most sophisticated editors would allow a "WYSIWYG" view and editing of an object representing some GUI element, such as a window. The Small App framework should be mature enough so that such editors can be written.

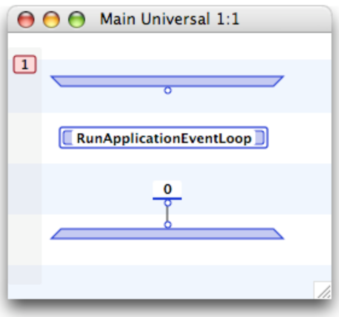
4. No NIB files - Since our goal is to be able to write our own editors, we won't need Interface Builder or some other resource editor to create resource files. Consequently, every "resource" such as a menu or a control will be created programmatically. This allows us to create a application entirely within the Marten IDE without the need of third-party tools.

5. Robust - While the framework will be limited in its power, it will still handle errors and allow the user to exit the application gracefully.

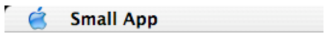
So that's the goal! Let's see if we can get there.

The simplest application

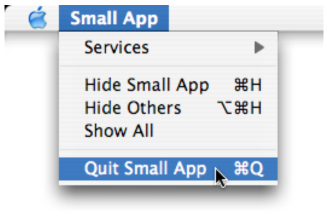
To get a feel for how simple a Macintosh application can be, create a universal that consists of the following code:



Install this universal as the MAIN method and run the application. The Small App application will come to the foreground and the menu bar will now display the system menu for Small App.

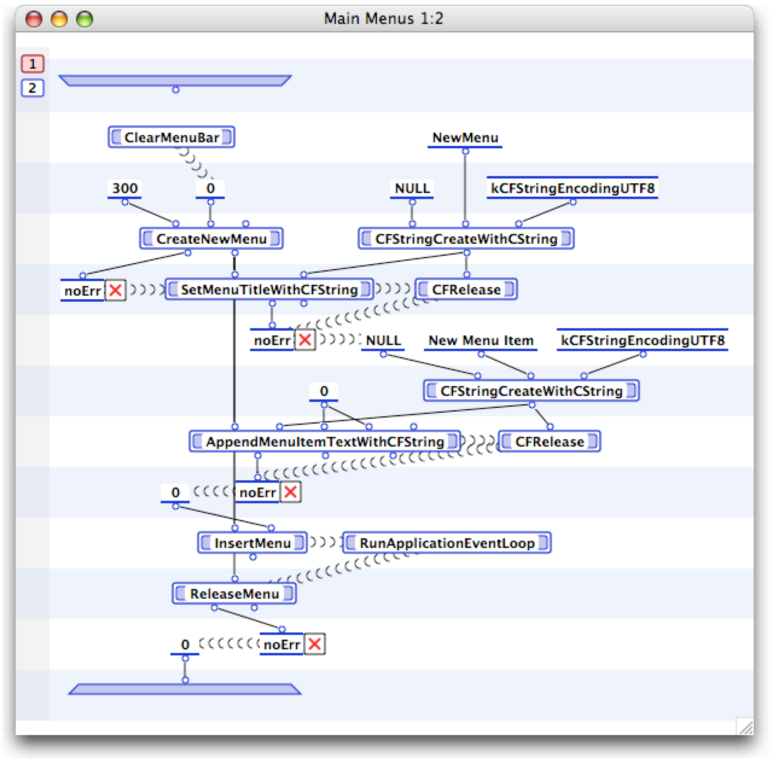


Now RunApplicationEventLoop is a system routine that clearly supplies a large amount of functionality. In the old days of MacOS, the developer would have been responsible for putting up the "system" menu and populating it with the Quit menu item. Then the developer would have had to write code that pulled events off the event queue, tested the events for kind (such as Mouse Down), and where (say in a menu bar), and then handled such events (like Quit). Just to implement the functionality illustrated above by Small App would take a few hundred lines of code. Obviously things have evolved substantially and MacOS X developers have been freed from having to write such large amounts of boilerplate code. To quit Small App, just mouse down on the system menu and select "Quit Small App".

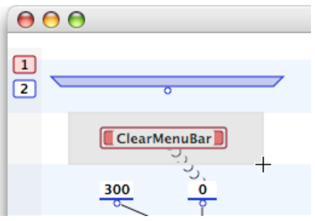


Menus

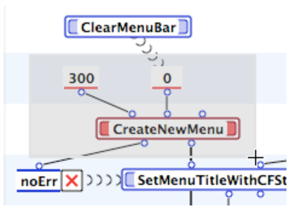
Now let's up the ante a little bit. Let's put a menu called "NewMenu" with a menu item entitled "New Menu Item" up onto the menu bar. We'll put the code for this into one large case so that we can see everything all at once.



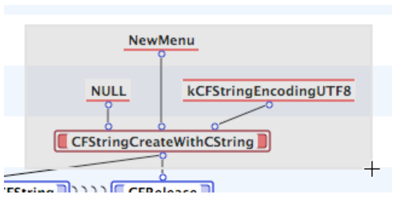
To see what is going on, we'll walk through it step by step. Because we are creating a menu programmatically, we must first call ClearMenuBar to give us a clean menu bar to work with.



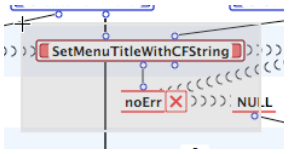
Next we create a new menu using the CreateNewMenu routine. This routine takes a menu identifier that we arbitrarily chose as 300 and a set of bits that control some attributes of the menu. We just want the defaults, so we pass 0 in at the second terminal.



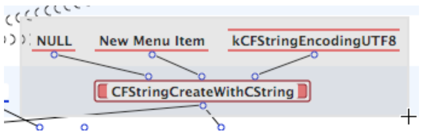
One difference this routine has with respect to the old "NewMenu" routine is there is no place to input the title of the menu. In the new API, we must set the title programmatically. Most of the modern MacOS X APIs require a Core Foundation string in leu of a C string. We can create such a "CFString" by using the procedure "CFStringCreateWithCString". This procedure requires an allocator which we set to NULL, the C string (which can be extracted automatically from the Marten string), the encoding of the array of bytes that comprises the C string. The native encoding for Marten strings is UTF8 so we pass the constant "kCFStringEncodingUTF8" into the routine. Notice (not depicted below, refer to the large diagram) that we must "release" the CFString after we are done passing it into the routine.



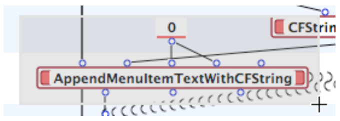
Now we set the title with "SetMenuTitleWithCFString".



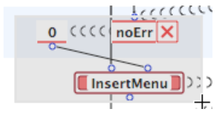
Now a MacOS X menu item is just some text stored in the menu at a certain offset. Remembering that we need a CFString, we create one for the menu item.



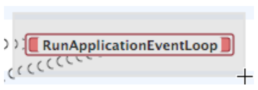
While there are procedures that allow us to place the menu item anywhere in the menu item list of the menu, we just add it at the end using the "AppendMenuItemTextWithCFString" routine.



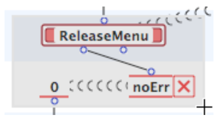
Finally we have to add the menu to the menu bar. If we pass 0 to the "InsertMenu" routine, it will do just that.



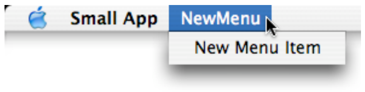
Since we are done with the creation of the menu bar, we are free to run the application.



After the Quit command has been issued and "RunApplicationEventLoop" has been exited, we must cleanup. Now since we will exit to the shell immediately, we could just let the system clean up. However, we want to handle the general situation so we explicitly call "ReleaseMenu" to dispose of our menu.



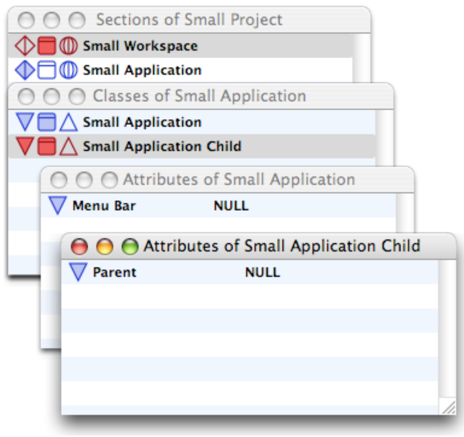
Install the method as the MAIN method and run the application. Our menu will show up to the right of the system menu and contain our one menu item.



Design of the classes

Given that we have the code for a complete application sitting in front of us, we can easily work out a design for a class framework encompassing an application and its menus. The first step is to recognize that the initial objects are an application and some elements (like menus) of an application. This framework will be hierarchical in structure; basically a tree with the root object being the application. So we start out with two classes, the application class "Small Application" and an abstract base class representing any element of an application "Small Application Child". The "Child" is deliberate, it emphasizes the "parent-child" relationship between the elements of this tree.

To get our framework started, we create a "Small Application" section to store the two classes. The sole attribute of the Small Application Child class is just "Parent", which will allow us to navigate to any node on the tree. The first attribute of the Small Application class is "Menu Bar" which we know we'll need to store the menu bar, whatever it will turn out to be.

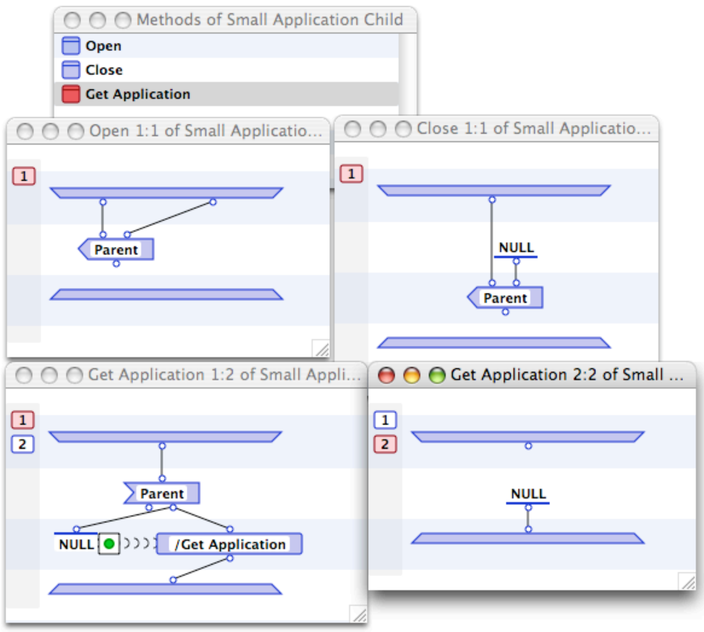

Sections of Small Project
Small Workspace
Small Application

Classes of Small Application
Small Application
Small Application Child

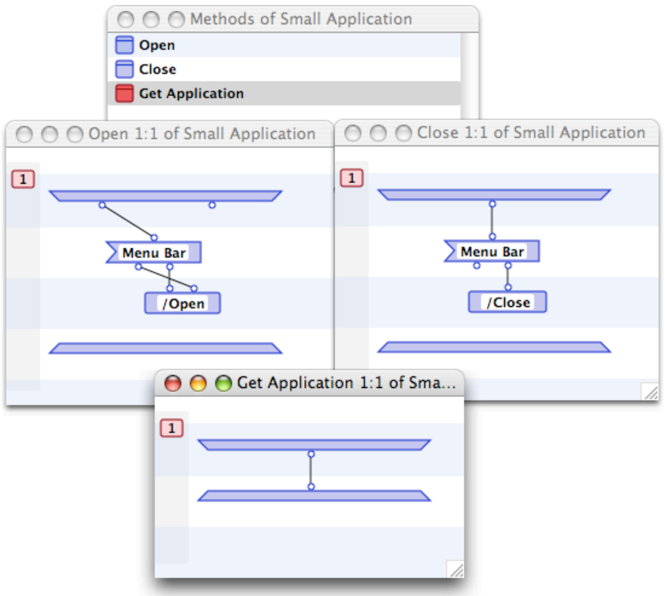
Attributes of Small Application
Menu Bar NULL

Attributes of Small Application Child
Parent NULL

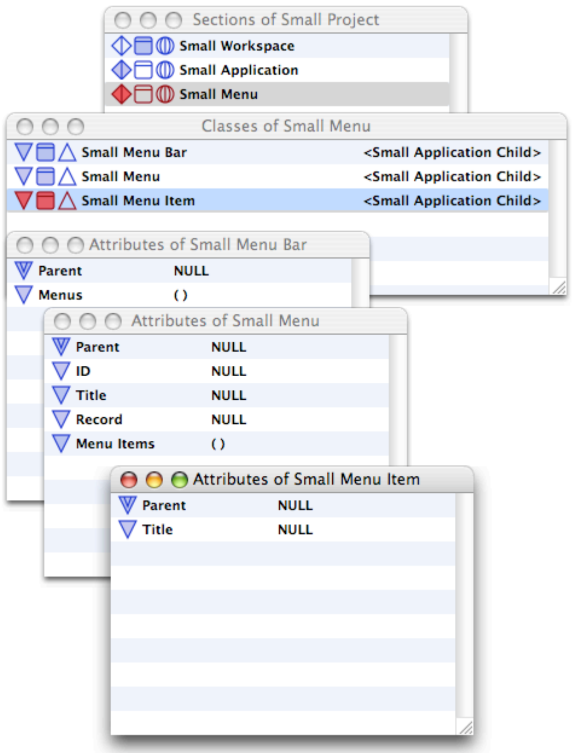
Now the key characteristic of the Small Application Child class is the ability to access the application instance, the root of the tree. In order to do this, each instance of a Small Application Child class must store a reference to its parent and the parent must be able to access the application instance itself. So we construct three methods, "Open" to store the parent, "Close" to delete the reference to the parent, and "Get Application" to access the application instance.


Methods of Small Application Child
Open
Close
Get Application
Open 1:1 of Small Applicatio...
1
Parent
Close 1:1 of Small Applicatio...
1
NULL
Parent
Get Application 1:2 of Small Appli...
1
2
Parent
NULL
/Get Application
Get Application 2:2 of Small ...
1
2
NULL

Now looking ahead, we know that we will have some kind of class that represents a menu bar. Consequently, the Small Application class must have an Open method which opens its sub-elements (the menu bar for now), a Close method that closes the sub-elements, and of course a Get Application method that just returns itself. Notice that we keep a consistent inarity of 2 for Open methods.


Methods of Small Application
Open
Close
Get Application
Open 1:1 of Small Application
1
Menu Bar
/Open
Close 1:1 of Small Application
1
Menu Bar
/Close
Get Application 1:1 of Sma...
1

Notice that we are not doing any "real" coding yet where we make calls to primitives or procedures. We are just getting the hierarchy organized, starting from a top level. So continuing in the same vein, we move on to the Small Menu section. We create a Small Menu section that contains three classes to represent menu bars, menus, and menu items. Again, the organization is hierarchical so that an instance of a Small Menu Bar contains a list of Small Menu instances which in turn contain a list of Small Menu Item instances. In addition, the menu code that we wrote earlier makes it clear that creation of a menu involves an ID and a title which yields a reference to system representation of a menu. So the Small Menu must also store the ID, the Title, and the Record. The addition of a menu item to a menu just requires the insertion of some text, so the Small Menu Item must also store the Title.



Sections of Small Project

- Small Workspace
- Small Application
- Small Menu

Classes of Small Menu

Small Menu Bar	<Small Application Child>
Small Menu	<Small Application Child>
Small Menu Item	<Small Application Child>

Attributes of Small Menu Bar

Parent	NULL
Menus	( )

Attributes of Small Menu

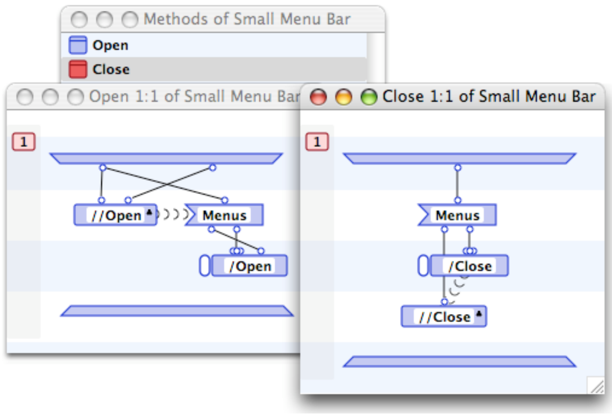
Parent	NULL
ID	NULL
Title	NULL
Record	NULL
Menu Items	( )

Attributes of Small Menu Item

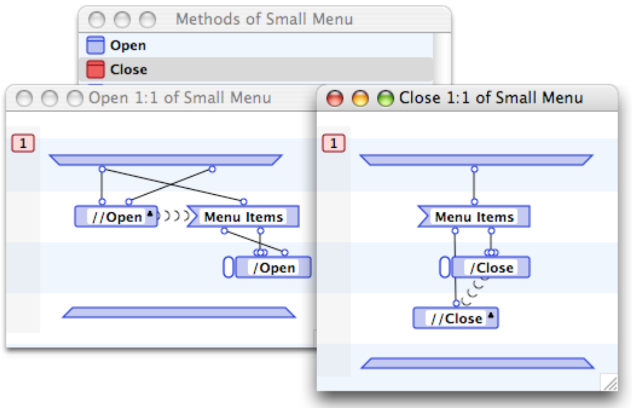
Parent	NULL
Title	NULL

The attributes recorded above are the minimal set required to capture all the necessary information.

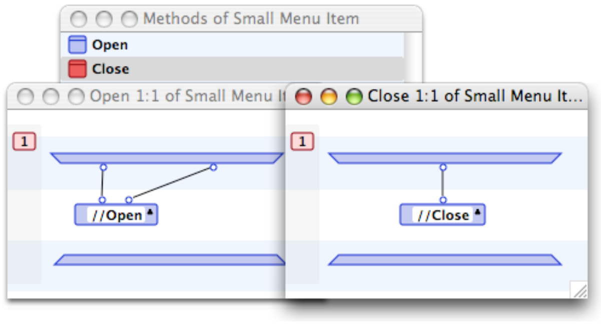
To code up the methods, we first take care of the hierarchy. We will come back and add additional code to handle the system calls later. So proceeding straightforwardly, a Small Menu bar must open itself first (with a call to its "super" class method) and then open all its menus. And when a Small Menu bar is closed, it closes its menus first and then closes itself.


Methods of Small Menu Bar
Open
Close
Open 1:1 of Small Menu Bar
1
//Open
Menus
/Open
Close 1:1 of Small Menu Bar
1
Menus
/Close
//Close

The Small Menu methods are exactly the same. When opened, an instance opens itself first and then its menu items and when closed, it closes its menu items first and then closes itself.

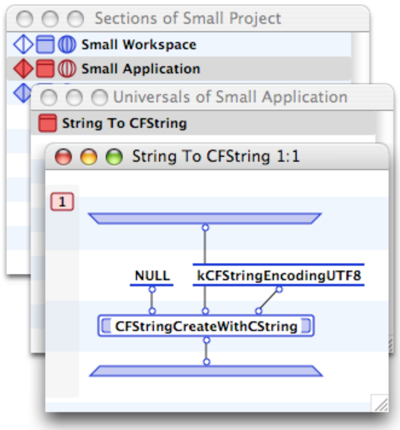


Finally, since a Small Menu Item is not a container for further sub-elements, it just opens and closes itself.

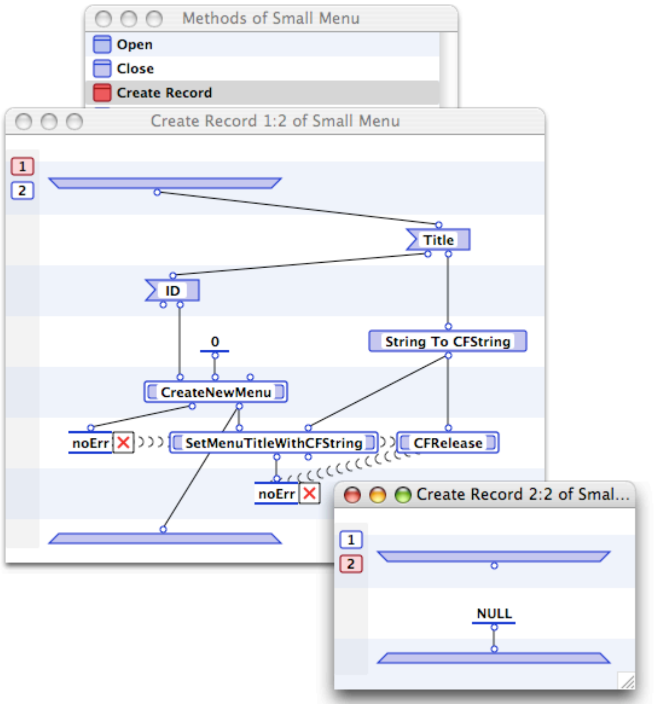

Methods of Small Menu Item
Open
Close
Open 1:1 of Small Menu It
1
//Open
Close 1:1 of Small Menu It...
1
//Close

The real code

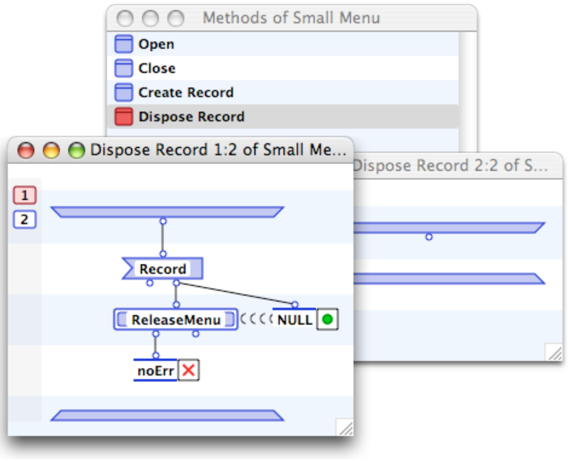
Now we get to work, writing code that involves the MacOS X Carbon Application Programming Interface (API). The first thing we will do is factor out the duplicated code that creates a Core Foundation (CF) string. We select the appropriate operations in our large test case and then issue the "Operations To Local" command. Then we send the "Local To Method" command, name the new universal method "String To CFString", and save it in the Small Application section.



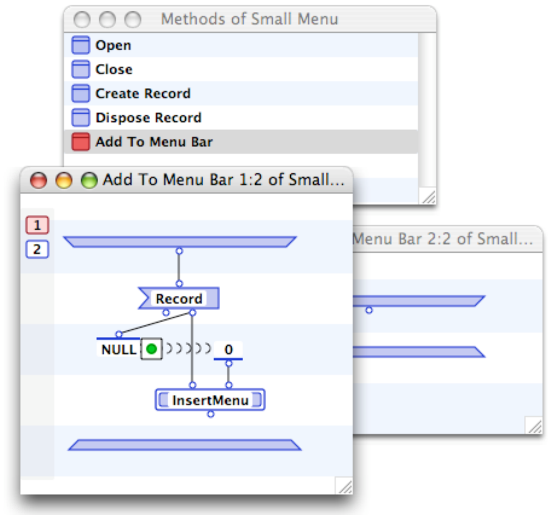
Since we already have working code, it is a simple matter of cut and paste to write the "Create Record" method of the Small Menu class. This method takes the stored ID and Title of a Small Menu instance and supplies them to the CreateNewMenu and SetMenuTitleWithCFString routines we described earlier:



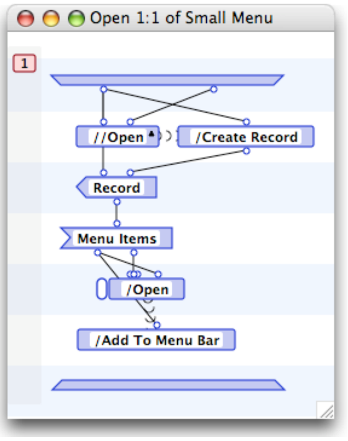
If we can create a menu record, we must be able to dispose of it as well. So we create a "Dispose Record" method and again, cut and paste the previous code into it.



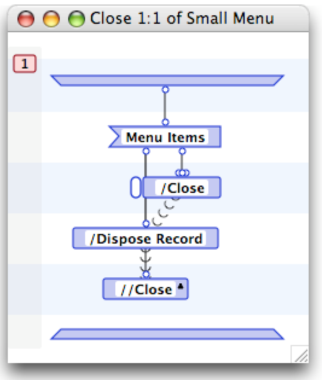
Finally we need a method to add the menu to the menu bar. We create an "Add To Menu Bar" method and copy the appropriate code into it. That completes the 5 methods of Small Menu.



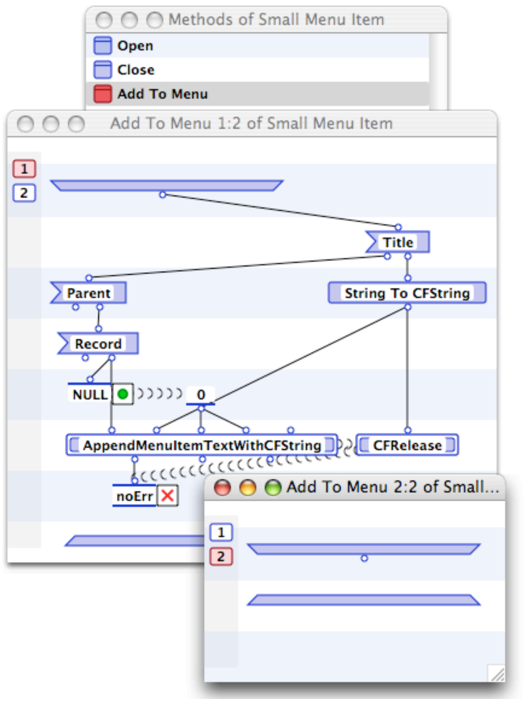
We now return to the Open and Close methods of the Small Menu class in order to update them with our "real routines". We make sure we create the menu record before we open the menu items and after we finish with them, we add the menu to the menu bar.



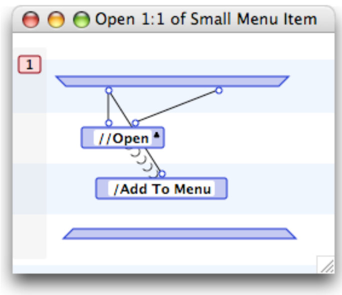
Similarly, we update the Close method to dispose of the menu record after we have closed the menu items but before we call the super method.



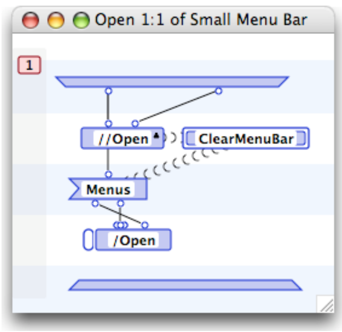
Moving on to the Small Menu Item class, we must write a routine that adds the menu item to the menu. We code up an "Add To Menu" method, copy the code into it, and hook the code up to the attributes of an incoming instance.


Methods of Small Menu Item
Open
Close
Add To Menu
Add To Menu 1:2 of Small Menu Item
1
2
Title
Parent
String To CFString
Record
NULL
0
AppendMenuItemTextWithCFString
CFRelease
noErr
Add To Menu 2:2 of Small...
1
2

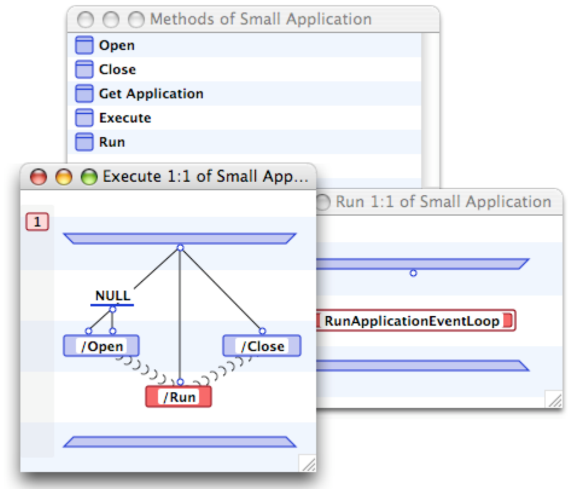
We update the Open method for the Small Menu Item class with a call to add the menu item to the menu. We have no need to remove the menu item yet so we postpone the creation of a "Remove From Menu" method and the consequent update of the Close method to a later chapter.



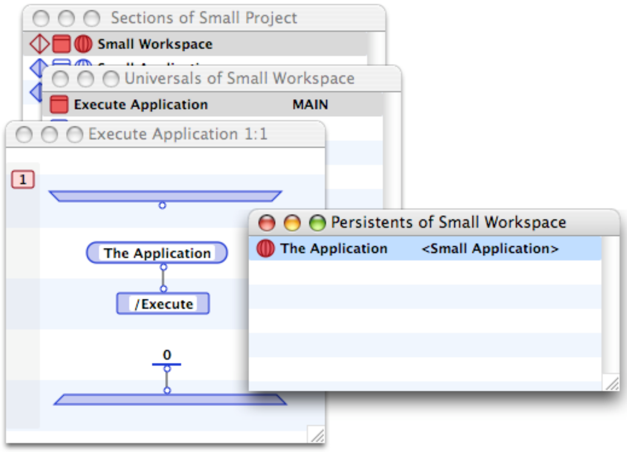
Finally we add just a little additional code to the Open method of the Small Menu Bar class to initialize the menu bar before opening the sub-elements.



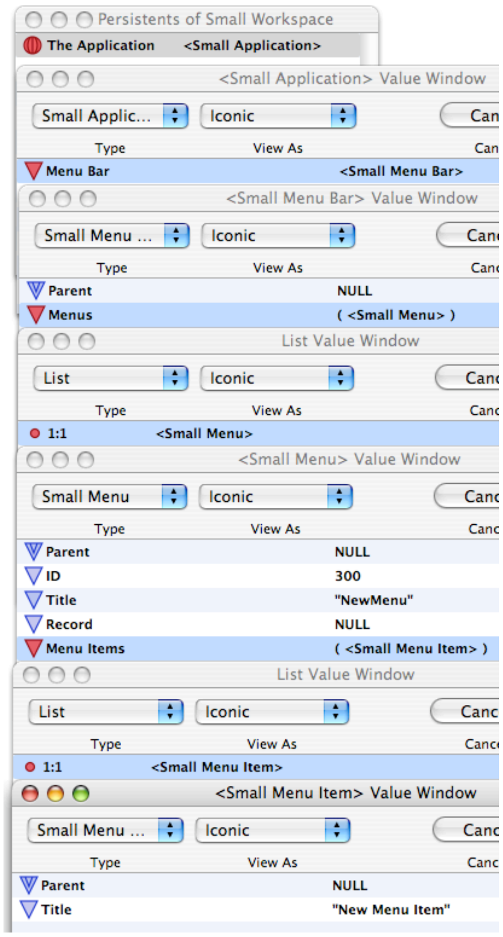
Turning back the the Small Application class, we see that in order to execute the application, we must open the application, run it, and then close it down. So we create two new methods, "Execute" and "Run". The Run method is a just simple placeholder for the RunApplicationEventLoop procedure.


Methods of Small Application
Open
Close
Get Application
Execute
Run
Execute 1:1 of Small App...
1
NULL
/Open
/Close
/Run
Run 1:1 of Small Application
RunApplicationEventLoop

We are just about good to go. There is just one last item to attend to and that is how do we kick the whole thing off? We need a MAIN method that will execute an instance of an application. We could have the method call the Instantiate operation for the Small Application class, but in keeping with the design philosophy of being able to edit instances, it is much better to create a persistent that can store an instance of the application and call it in the MAIN method. This persistent should be stored in a user-modifiable section, a "workspace" that allows the basic framework to remain untouched. This way we can edit the specific instance of our application rather than changing the default attributes of the Small Application class. So we create a persistent called "The Application" and using the value editors, store an instance of the Small Application class in it. And we write a universal method called "Execute Application" to grab that instance and supply it to the "Execute" method. This method is installed as the MAIN method of the project. This persistent and universal are saved in the "Small Workspace" section.



Now we open up a set of Value Editors to install various instances of Small Menu Bars, Small Menus, and Small Menu Items to recreate the functionality that we initially provided in just the one case.


Persistents of Small Workspace
The Application <Small Application>

<Small Application> Value Window
Small Applic... Iconic Can
Type View As Can
Menu Bar <Small Menu Bar>

<Small Menu Bar> Value Window
Small Menu ... Iconic Canc
Type View As Canc
Parent NULL
Menus (<Small Menu>)

List Value Window
List Iconic Canc
Type View As Canc
1:1 <Small Menu>

<Small Menu> Value Window
Small Menu Iconic Canc
Type View As Canc
Parent NULL
ID 300
Title "NewMenu"
Record NULL
Menu Items (<Small Menu Item>)

List Value Window
List Iconic Canc
Type View As Canc
1:1 <Small Menu Item>

<Small Menu Item> Value Window
Small Menu ... Iconic Canc
Type View As Canc
Parent NULL
Title "New Menu Item"

Closing up the editors creates an instance of the Small Application class that when executed puts up the same menu and menu item that we had before. The essential difference is that no more *coding* has to be done to put up any combination of menus and menu items. All we have to do is *edit* the application instance appropriately. Give it a try. One good project is to try to recreate the menus and menu items of a favorite program, such as the Address Book application or Safari. You will soon see that we need a bit more to recreate them faithfully (such as separators, command key equivalents, and submenus) but we are well on our way.

Next on the agenda is to provide the menu items with a way to invoke a command so that we can get some work done. This will involve creating a framework for the Carbon Event Model. We will associate menu items with commands and provide callbacks to translate those commands into execution of our methods. This will involve adding more attributes to our classes and coding up more methods, but for now, the current class structure is described below.

Universals Reference

Methods:
- String To CFString
  - Inputs: String
  - Outputs: MacOS X CFString
  - Description: Creates a CFString from a Marten string. The CFString must be disposed of with a call to CFRelease.

Class Reference

Small Application
Description: This class represents the application; a container for all menus, windows, event handlers, etc.

Attributes:
- Menu Bar
  - Default: NULL

Description: A slot for a Small Menu Bar instance - MUST be non-null when Execute is called.

Methods:
- Open
  - Inputs: Instance, NULL
  - Outputs: No outputs
  - Description: Calls the open methods of all sub-elements.
- Close
  - Inputs: Instance
  - Outputs: No outputs
  - Description: Calls the close methods of all sub-elements.
- Get Application
  - Inputs: Instance
  - Outputs: Instance
  - Description: Returns self.
- Execute
  - Inputs: Instance
  - Outputs: No outputs
  - Description: Calls Open, Run, then Close.
- Run
  - Inputs: Instance
  - Outputs: No outputs
  - Description: Calls the Carbon system routine RunApplicationEventLoop.

Small Application Child
Description: A abstract base class for all sub-elements of an application.

Attributes:
- Parent
  - Default: NULL
  - Description: A slot for the container of the sub-element.

Methods:
- Open
  - Inputs: Instance, Parent container
  - Outputs: No outputs
  - Description: Sets the Parent attribute to the container.
- Close
  - Inputs: Instance
  - Outputs: No outputs
  - Description: Sets the Parent attribute to NULL.
- Get Application
  - Inputs: Instance
  - Outputs: Small Application Instance
  - Description: Calls Get Application on the parent, which in turn calls Get Application and so forth. This set of calls "travels" up the hierarchy tree until the root application is reached, which is then returned as the output.

Small Menu Bar
Description: An application sub-element that represents a menu bar.

Attributes:
- Parent
  - Default: NULL
  - Description: A slot for the parent Small Application.
- Menus
  - Default: Empty List
  - Description: A list of Small Menu instances.

Methods:
- Open
  - Inputs: Instance, Small Application Instance
  - Outputs: No outputs
  - Description: Calls the Open super method, clears the menu bar, and then finally opens all menus.
- Close

Inputs: Instance
Outputs: No outputs
Description: Closes all menus and then calls Close super method.

Small Menu
Description: An application sub-element that represents a menu.

Attributes:
- Parent
  - Default: NULL
  - Description: A slot for the parent Small Menu Bar.
- ID
  - Default: 0
  - Description: An integer representing the menu identifier.
- Title
  - Default: NULL
  - Description: A string containing the text of the menu.
- Record
  - Default: NULL
  - Description: A slot for the MacOS X MenuRef representing the menu.
- Menu Items
  - Default: Empty List
  - Description: A list of Small Menu Item instances.

Methods:
- Open
  - Inputs: Instance, Small Menu Bar Instance
  - Outputs: No outputs
  - Description: Calls the Open super method, creates and stores the menu record, opens the menu items, and then finally adds self to the menu bar.
- Close
  - Inputs: Instance
  - Outputs: No outputs
  - Description: Closes all menu items, disposes of the menu record, and then calls Close super method.
- Create Record
  - Inputs: Instance
  - Outputs: MacOS X MenuRef
  - Description: Using the ID, creates a menu record and then sets the menu title.
- Dispose Record
  - Inputs: Instance
  - Outputs: No outputs
  - Description: Disposes of the menu record.
- Add To Menu Bar
  - Inputs: Instance
  - Outputs: No outputs
  - Description: Adds itself to the current menu bar.

Small Menu Item
Description: An application sub-element that represents a menu item.

Attributes:
- Parent
  - Default: NULL
  - Description: A slot for the parent Small Menu.
- Title
  - Default: NULL
  - Description: A string containing the text of the menu item.

Methods:
- Open
  - Inputs: Instance, Small Menu Instance
  - Outputs: No outputs
  - Description: Calls the Open super method and then adds self to the menu bar.
- Close

Inputs: Instance
Outputs: No outputs
Description: Calls Close super method.

Add To Menu

Inputs: Instance
Outputs: No outputs
Description: Adds itself to the current menu.