;ToDo_Intro.rtfd;linkMarkername ;¬ Previous Section ;B_ToDo_SettingUp.rtfd;linkMarkername ;¬ Next Section

4. To Do Tutorial

The Design of To Do

The To Do application vaults past Travel Advisor in terms of complexity. Instead of Travel Advisor's one nib file, To Do has three nib files. Instead of three custom classes, To Do has seven. This diagram shows the interrelationships among instances of some of those classes and the nib files that they load:

TD_Design1.eps ¬

Some of the objects in this diagram are familiar, fitting as they do into the Model-View-Controller paradigm. The ToDoItem class provides the model objects for the application; instances of this class encapsulate the data associated with the items appearing in documents. They also offer functions for computing subsets of that data. And then there's the controller object...actually, there is more than one controller object.

> The ToDoInspector instance in the above diagram is an offshoot of the application controller, ToDoController. By breaking down a problem domain into distinct areas of responsibility, and assigning certain types of objects to each area, you increase the modularity and reusability of the object, and make maintenance and trouble-shooting easier. See ªObject-Oriented Programmingº in the appendix for more on this.

To Do's Multi-Document Design

Two types of controller objects are at the heart of multi-document application design. They claim different areas of responsibility within an application. ToDoController is the *application controller*; it manages events that affect the application as a whole. Each ToDoDoc object is a *document controller*, and manages a single document, including all the ToDoItems that belong to the document. Naturally, it's essential that the application

controller be able to communicate with its (potentially) numerous document controllers, and they with it.

As multi-document applications typically do, To Do includes the Document menu found on Interface Builder's Menus palette. When users choose New from the Document menu, the application controller allocates and initializes an instance of the ToDoDoc class. When the ToDoDoc instance initializes itself, it loads the **ToDoDoc.nib** file. When the user has finished entering items into the document, and chooses Save from the Document menu, a Save panel appears and the user saves the document in the file system under an assigned name. Later, the user can open the document using the Open menu command, which causes the Open panel to be displayed.

> The rationale behind, and process of, constructing multi-document applications is discussed in ``The Structure of Multi-Document Applications.'' ;ToDoConcepts.rtfd;linkMarkername TheStructureofMulti-DocumentApplications;¬

The controller objects of To Do respond to a variety of delegation messages sent when certain events occurÐprimarily from windows and NSAppÐin order to save and store object state. One example of such an event is when the user closes a document window; another is when data is entered into a document. Often when these events happen, one controller sends a message to the other controller to keep it informed.

How To Do Stores and Accesses its Data

The data elements of a To Do document (ToDoDoc) are ToDoItems. When a user enters an item in a document's list, the ToDoDoc creates a ToDoItem and inserts that object in a mutable array (NSMutableArray); the ToDoItem occupies the same position in the array as the item in the matrix's text field. This positional correspondence of objects in the array and items in the matrix is an essential part of the design. For instance, when users delete the first entry in the document's list, the document removes the corresponding ToDoItem (at index 0) from the array.

TD_Design3.eps ¬

The array of ToDoItems is associated with a particular day. Thus the data for a document consists of a (mutable) dictionary with arrays of ToDoItems for values and dates for keys.

TD_Design4.eps ¬

When users select a day in the calendar, the application computes the date, which it then uses as the key to locate an array of ToDoItems in the dictionary.

To Do's Custom Views

The discussion so far has touched on model objects and controller objects, but has said nothing about the second member of the Model-View-Controller triad: view objects. Unlike Travel Advisor, which uses only ªoff-the-shelfº views, To Do's interface features objects from three custom Application Kit subclasses.

TD_Design5.eps ¬

You'll learn much more about these custom subclasses in the pages that follow.