9.   The Info menu

The GameKit includes an object call the InfoController which handles things like help, registration, order forms, mailing suggestions to the game's author, and the Info¼ panel itself.   It also has a simple View subclass to aid in creating animated Info¼ panels known as the AnimatedView.   *It might make more sense for these object to be in the DAYMiscKit, since they really are generic, and not specific to a game application.   Also, I will probably add a class to this for handling registration, so that the InfoController doesn't actually need to be subclassed to handle registration.   Instead, just connect it to an object that can say yay or nay.*

InfoController

The main function of the InfoController object is to sit underneath the Info menu.   Clicking a button on the Info menu will typically send an action message to the InfoController, which will then perform the desired function.   In many cases, this is to simply bring up a panel.   In other cases, it brings up a particular help file in the Help panel.   (Note that the Help¼ menu item should be connected as is standard for NEXTSTEP 3.x±to bring up the 3.x help panel.)   The following files are used to implement the various InfoController features; your app should include them as described below if you wish to take advantage of the features which they implement:

InfoPanel.nib	Contains the application's Info¼ panel. It should be part of the English.lproj for your application. This panel might contain special animated views as described in the next section.
Register.nib	Contains a panel which may be used to enter registration keys. This panel is set up by the InfoController to be a modal panel. See the section below on registration for more information.
OrderForm.nib	Contains a panel which may be used as an order form for your application. There are InfoController methods to help do rudimentary price extension for multiple copies.
README.rtfd	A file which should be part of the NEXTSTEP Help for your application. This file is brought up the first time your application is launched and should be treated like release notes, or any sort of ªREADMEº file. By putting it in the application and forcing the user to see it at least once, there is a greater chance that user will actually read some of what you put in here. (Put the important stuff first, of course, so that it will be read before the user loses interest in the panel!) Sending the InfoController a ±**readme:** message brings up the Help¼ panel with this file showing.
License.rtfd	Another file which should be in the Help directory. This file should contain complete details of the software license. This file can be brought up from an appropriate menu item, so a user can quickly locate the license for you application. Sending the InfoController a ±**license:** message brings up the

Help¼ panel with this file showing.

The following sections detail the features which the .nib files listed above implement. Before that, however, there is one other feature available±sending a suggestion to the game's author. This is done by sending a message to Mail.app and loading an appropriate skeleton message into a Compose¼ window and happens when the InfoController gets a ±**suggestion:** message. If you want to change the e-mail address, subject line, or the skeleton message, all of them are in the main string table of the application and may be adjusted from within the string table itself. *I need to update the method used so that it doesn't smash the contents of any open Compose¼ windows. This will eventually be fixed. For now, there is an alert panel to warn the user.*

AnimatedView ± making impressive Info¼ panels

The Info¼ panel should be inside of its own .nib file, InfoPanel.nib, which has an InfoController as the file's owner. The panel should be connected to the **infoPanel** outlet of the InfoController. The InfoController will load the .nib file, if necessary, and then bring up this panel when an ±**info:** message is received. The ±**infoPanel** method returns the **id** of the Info¼ panel's window, loading it from the .nib file if necessary. On the panel itself, there should be two TextFields which are connected to the **versionText** and **versionDateText** outlets of the InfoController. When the Info¼ panel is ordered out, these fields with be filled with appropriate values taken from the GameKit's global string table, which is obtained from the GameBrain object. Make sure that the text fields are set up so that the version number and date will look alright when filled in.

Another connection which may be made, but is optional, is to put a subclass of AnimatedView in your Info¼ panel. This View subclass is designed to provide simple animations to make Info¼ panels more interesting. If such a View subclass is connected to the InfoController's **niftyView** outlet, then it will be sent a ±**start:** message when the Info¼ panel is displayed. Make sure that the AnimatedView is set up to be the delegate of the Info¼ panel's window so that it will stop animating if the panel is closed.

The animated view does two things: it sets up a timed entry (via an Animator object) to call itself periodically, and it handles starting and stopping animation at the appropriate times. Your subclass needs to implement the animation itself. To do this, override the ±**autoUpdate** method to draw the next frame of animation. This method will be called once per frame. If you need to do any special initializations or clean ups, you might need to override ±**stop:**, ±**start:**, **±free**, and ±**initFrame:** as necessary. Be sure to call the **super** implementation if you override any of these methods. When you override the ±**autoUpdate** method, be sure to bracket all your drawing with lock and unlock focus messages to **self**. *Right now, there are no publically available examples of an AnimatedView subclass. I might put one into PacMan or NX_Invaders eventually. If you are stck on how to implement an ±**autoUpdate:** method, Columns does have an actual example of this.*

Handling registration keys

The Register¼ panel is found inside the Register.nib file. The InfoController outlet **registerPanel** should be connected to the Register¼ panel. There should also be two TextFields, connected to the InfoController outlets **regText** and **regNumText** on the panel. The **regText** field should be uneditable and will be loaded with a serial number, taken from the **serialNum** instance variable. (You put something in the **serialNum** variable, as explained below.) The **regNumText** is for the user to enter the registration key. There should be two buttons on the panel, one to register and one to cancel the operation. They connect to the InfoController's ±**registerGame:** and ±**cancel Registration:** methods, respectively. Once all these connections are made, all that is left is for you to provide some way to check for valid keys and provide serial numbers.

There is a single method which you must override in order to provide for a registration mechanism. This is the ±**keyOK** method. In this method, you must do two things. First determine from the key, found in the **key** instance variable, a unique serial number. The serial number should be copied into the **serialNum** instance variable. Next, determine if the key string is valid or not. If not, then return NO. Return a YES if it is acceptable. By default, if the key is invalid, the serial number ªNREGº will be used regardless of the vaule of the **serialNum** instance variable. The serial number itself is displayed in the registration panel and the subject line of the mail message sent to the author.

No actual mechanism is provided to validate keys since each vendor will want to use their own amazing, proprietary, uncrackeable method. Note that any good hacker who understands the GameKit could get past this whole mechanism anyway, as well as any other thing you might come up with, GameKit related or not. This will only provide a very simple level of security. Doing much more is probably not worth the effort since the enterprising *will* figure it out; the more challenging you make it, the more likely they'll want to break it just to say that they did it!

Making an order form

To build an order form, you will have to do a little bit of work. First, lay out your order form in InterfaceBuilder. The window should be in a separate .nib file (OrderForm.nib) with the InfoController as the file's owner. The window should be connected to the **orderFormPanel** outlet of the InfoController. Note that the panel will be prineted on a page by printing the whole window. There is a Print button on the panel, which sends a ±**printOrderForm:** message to the InfoController. Connect the **costText** outlet of the InfoController to a TextField which will display the total sum of money to send to the author.

For you make make your order from, you must add outlets to your InfoController subclass to connect to the various fields which specify things like how many licenses are ordered and so on. Then, override the ±**costCalc:** method to take the values from the fields you added, calculate the cost, and then set the value of the **costText** TextField to the sum you have calculated.   You may wish to have the last text field in the chain send a ±**costCalc:** message to the InfoController, but this is not necessary since the method will always be invoked before printing anyway.   The default ±**costCalc:** simply puts a 10 in the field; you will most likely want to change this to suit your pricing structure.

If your game is free, then simply omit this .nib file and don't have an Order Form¼ entry in the Info menu.   Leaving this section unconnected will not harm anything else.   The same goes for the registration mechanism.   Both are entirely optional when using the InfoController object.