CS193E: Assignment 3
FavoriteThings II

Due Date

This assignment is due by **11:59 PM, January 30**.

Assignment

This week's assignment builds on the Favorite Things application you built for Assignment 2.

This week you will extend the application, add a number of user interface refinements and add a preferences pane.

Here's a rough outline of what you'll need to do to complete the assignment successfully:

Extending the model and view

1. Add a favorite color. There should be an NSColor stored in the model, and the favorite color should be displayed using an NSColorWell in the view.
2. Add a favorite picture. There should be an NSImage stored in the model, and it should be displayed using an NSImageView in the view.
3. Allow user to select favorite picture. Add a button next to the image view that presents the open panel to allow the user to select their favorite picture. The open panel should only allow the user to select valid image types. The user should only be able to choose one image at a time.
4. The title and button prompt of the open panel should be customized before it is displayed to be more appropriate to the operation being performed, since from the user's perspective we are not opening a file, but selecting a favorite image.

Other Enhancements

5. Remove the "Save" button, and make the existing "Save" menu item in the Edit menu work instead. The Save menu item should only be enabled if there are unsaved changes in the window.
6. If there are unsaved changes when user tries to close the window, present an alert sheet asking them if they'd like to save changes, cancel, or don't save changes. Save changes should save the changes and close the window, don't save changes should not save changes and close the window. Cancel should not save changes and leave the window open.
7. Set the bundle identifier for the application. The bundle identifier should contain the class number and your Stanford user name. (e.g. edu.stanford.cs193e.dempsey1.FavoriteThings).
8. Add the provided icon file as a resource in your project and set it as the application icon. When adding the icon to your Xcode project, be sure to either first put the icon inside of the directory containing your source code or check the "Copy items into destination group's folder (if needed)" checkbox on the alert shown in Xcode when adding the file to the project. Remember, only files inside of your source directory are submitted with the submission script.

Preferences

9. Add a preferences window. The preferences window should be in its own nib file and be loaded by your custom subclass of NSWindowController. Hook up the existing Preferences… menu item so that choosing it opens the Preferences window.
10. Move the "Quit when window closed" checkbox to the preferences window.
11. Store and retrieve the preference in user defaults.

Testing

Make sure the following work in your application (these are the things we will test to determine your grade):

0. Your project should build without errors or warnings.
1. Your project should run without crashing.
2. All things from Favorite Things I should continue to work
3. The new favorite things should behave the same as the existing ones with regards to editing, saving, and loading.
4. Should be able to click button to select an image. This should bring up an open panel. The open panel should:
    - Only allow image files to be selected
    - Only allow one item to be selected
    - Have a customized title and button prompt to reflect that the user is selecting not opening

5. Trying to close the window with unsaved changes should bring up an alert sheet. The sheet should have three buttons "Save", "Cancel", and "Don't Save".
    - "Save" should save changes and close the window.
    - "Cancel" should not save changes and leave the window on the screen
    - "Don't Save" should not save the changes and close the window.

6. Should be able to use the Save menu item when the document is edited. The Save menu item should only be enabled if there are unsaved changes in the window.
7. The application's bundle identifier should be properly set.
8. The application should have a custom icon.
9. The Preferences… menu item should bring up a preferences window. The "Quit when window closed" checkbox should be in the preferences window. The preference should persist between application launches. The application should change its behavior appropriately based on the preference.
10. We will verify that the preferences window is in its own nib file.
11. The preference should be stored in user defaults, not in the favorite things archive itself. You can test this as well as your bundle identifier using the `defaults` command line tool. (See Hints section for details)

Hints

NSOpenPanel is a subclass of NSSavePanel. A number of useful methods, including setting the prompt and title are declared in NSSavePanel.

For testing, there are a number of images located at /Library/Desktop Pictures.

Using an alert as a sheet is a two part process. First, you set up the alert and run it as a sheet. This includes providing a "modal delegate" and selector. Essentially, an object and a callback method that is called when the user is done with the sheet. In the second part, you implement the callback method, inspect the result code of the sheet, and take the appropriate action.

Remember there is a difference between the NSWindow methods `-close` (no argument) and `-close:` (takes an argument).

Usually the Application controller will manage the controllers for things like the preferences window. The application controller has a reference to the preferences controller and

implements an action method to show the preferences controller. When that method is called, if the preferences controller is not already created, it is created and displays the window.

Note that the preferences controller is the only controller that needs to know about the "Quit when window closed" checkbox. In the preferences controller changes to the checkbox get set in the standard user defaults. In the application controller, when that default needs to be checked, get the information from the standard user defaults.

You can read the user defaults of an application from the command line using the `defaults` command. (You can do even more with this tool, see the man page for details)

For an application with bundle identifier 'edu.stanford.cs193e.dempsey.FavoriteThings':

```
defaults read edu.stanford.cs193e.dempsey.FavoriteThings
```

will return the keys and values of all preferences set for that application. You can use this to verify that your preference is being set and that you have correctly set your bundle identifier.

Troubleshooting

Some controls such as NSColorWell throw an exception when a nil value is set. This can be avoided in a new FavoritesRepository object by starting with a default color value. However, remember that an existing archive will not contain a color. In initWithCoder: you may want to check to ensure that the value decoded from the archive is not nil. If it is nil, you can provide a default color value.

Extra Credit

Keep in mind that extra credit only applies if the required behavior outlined above is working. Don't spend time on extra credit until you've got the basics down.

- Æsthetics matter: a particularly attractive, innovative, or clever design of the interface is always appreciated. (If you want consideration for extra credit in this case, you must let us know when you submit your project.)
- In the sample applications posted on the website (and hopefully in your submissions as well), you may notice that the window is not marked as edited until after the user tabs out of a text field. Use notifications to find out that the text of a control changed, and mark the window as edited when this happens. Hint: NSTextField's superclass NSControl are where these notifications are declared and described.
- Window resizing. When you resize your FavoriteThings window, you will probably not get the same results as in the sample application. Set up autosizing in Interface Builder so that the views in your user interface (including the image view) size in a reasonable manner when the window is resized.

  Hint: Cocoa views can be set to automatically resize based on changes in the size of its superview. You can set how a view autosizes in Interface Builder using what are commonly called 'springs' and 'struts'. In Interface Builder, open Interface Builder Help from the Help menu and search for 'autosizing' for more details.
- Add any of the extra credit items from the previous assignment if you had not done so already. (Including the NSURL favorite thing and adding an entry to the Window menu)