UW-PCE-Apple-Summer-2014-Online / **HW4-NotableCollection-Writeup** PRIVATE

branch: master **HW4-NotableCollection-Writeup** / **HW4-Writeup.md**

MartinJNash 11 days ago Added section in writeup about creating new menu items and nil-target…

1 contributor

111 lines (73 sloc) 4.788 kb Raw Blame History

This is another document-based application. Each window will be a master-detail view with a collection of items on the left, and the individual slected item represented on the right.

General

- Use properties, not instance variables.
- Prefer `self.property` to `_property`.
- Name your assignemnt HW4_*studentid* so that we can easiliy tell whose work we are reviewing.

Overview

- Create a document based app.
- Create an `NSWindowController` subclass. Change its `init` method to always load a xib of the same name, as per the Mike Ash article.
- Delete the `MyDocument.xib` default xib file.
- Change the document.m file
  - keep a reference (property) to an instance of your new window controller.
  - delete `windowNibName` since you will be using your own NSWindowController which has its own xib.
  - delete `windowControllerDidLoadNib:`. Your window controller subclass is responsible for setting up its own view.
  - override `makeWindowControllers` to add your custom window controller. This is the place to alloc init your window controller. Make sure to call `self addWindowController:` to associate the doucment and window controller.

Window Controller

- add an `NSTableView` to the xib. It should be **view-based**, not cell-based. You can change this in the attributes inspector. Make sure you are inspecting the TableView, not its enclosing scroll view. This is a common issue.
- Create a prototype table cell view
  - Resize it to your liking
  - It should have an image view and a text field
- be the table view's data source and delegate
- manages a TodoList full of TodoItems or other aptly named collection of items (opportunity for code reuse!). You may use the classes I have provided as a starting point.

Here is a sampling of methods that you might want to include. Your implementation may have a different number.

```
+ (instancetype)notableCollectionWindowControllerWithList:(PCETodoList*)list
- (id)init
- (void)windowDidLoad

// NSTextFieldDelegate
- (void)controlTextDidChange:(NSNotification *)obj

// actions
- (IBAction)addItemButtonPushed:(id)sender
- (IBAction)removeItemButtonPushed:(id)sender
- (IBAction)imageDidChange:(id)sender

// items
- (void)updateInterfaceForItem:(PCETodoItem*)item
- (PCETodoItem*)currentlySelectedItem
- (PCETodoItem*)itemAtRow:(NSInteger)row
- (NSInteger)rowForItem:(PCETodoItem*)item

// table view methods
-(void)tableViewSelectionDidChange:(NSNotification *)notification
-(NSInteger)numberOfRowsInTableView:(NSTableView *)tableView
-(NSView *)tableView:(NSTableView *)tableView viewForTableColumn:(NSTableColumn *)tableColumn row:(NSInteger)row
```

Data model

- A collection of items. I used a slightly modified version of the TodoList model items. You may do the same.
- Add NSCoding compliance to your data model.
- Add an image property to your data model.

NSTableViewDelegate

Let the talbe view know how many rows to display.

```
-(NSInteger)numberOfRowsInTableView:(NSTableView *)tableView {
    // return number of rows based on the data you are displaying
}
```

Create an NSTableCellView from the example cell in the xib. Then populate it with data based on your data model.

```
-(NSView *)tableView:(NSTableView *)tableView viewForTableColumn:(NSTableColumn *)tableColumn row:(NSInteger)row
    // look up real name of method
    NSTableCellView *theCell = [tableView methodThatGetsYouATableCellViewBasedOnAStringIdentifier: @"identifier"]
    // set text and image
    ...
    return theCell;
}
```

Update your view when the user selects a new row.

```
-(void)tableViewSelectionDidChange:(NSNotification *)notification {
    // do some work
}
```

Main Menu

Create two new menu items in main.xib. You may add them to an existing tab, or create a new menu item.

Create two IBAction methods in your window controller. They must have the method signature `-(IBAction)methodName:(id)sender`. Of course you'll replace the name with your own. Populate your UI with dummy data in these two methods.

Back in the xib with a main menu. Drag from your newly created items to the 'first responder' item. A list of methods should pop up. Find your new methods in that list, and connect the actions.

First responder is the ui element that is currently selected or being used. If the first responder cannot respond, it's next responder is given a chance to respond, so on until there is no next responder. This is the responder chain, and your NSWindowController will have a chance to respond to the sent action.

This is called a nil-targeted action.

Considerations

- Your document and your window controller should have a reference to the same data object.
- You should save the document when closing its window.
- Make sure your table cell view prototype has an identifier. Set this in interface builder.