

---

# Outline View Programming Topics

User Experience: Data Presentation



2010-03-24



Apple Inc.  
© 2001, 2010 Apple Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.  
1 Infinite Loop  
Cupertino, CA 95014  
408-996-1010

Apple, the Apple logo, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

**Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY**

**DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

---

## **Introduction to Outline Views 7**

Who Should Read This Document 7

Organization of This Document 7

---

## **About Outline Views 9**

Behavior Inherited from NSTableView 10

---

## **Writing an Outline View Data Source 11**

Data Source Requirements 11

The Data Source and Memory Management 11

Sample Data Source Implementation 12

---

## **Using an Outline View Delegate 15**

---

## **Document Revision History 17**

---



# Listings

## Writing an Outline View Data Source 11

---

- Listing 1 Implementation of Outline View Data Source 12
- Listing 2 Implementation of Outline View Data Source Item 12



# Introduction to Outline Views

---

An outline view is a type of table which lets the user expand or collapse rows that contain hierarchical data.

## Who Should Read This Document

`NSOutlineView`, the class that implements outline views, is a subclass of `NSTableView`. Before reading about outline views, read *Table View Programming Guide*. In particular, The Parts of a Table describes how all the different classes used by a table fit together.

## Organization of This Document

This programming topic contains the following articles:

- [“About Outline Views”](#) (page 9) gives basic information on outline views.
- [“Using an Outline View Delegate”](#) (page 15) describes delegate methods for a outline view.
- [“Writing an Outline View Data Source”](#) (page 11) describes data source methods for an outline view.





# About Outline Views

`NSOutlineView` is a subclass of `NSTableView` that lets the user expand or collapse rows that contain hierarchical data. As in a table view, an outline view displays data for a set of related items, with rows representing individual items and columns representing the attributes of those items. Unlike a table view, items in an outline view are not in a flat list, but rather may be organized in a hierarchy, like files and folders on a hard drive, or managers and employees in an organization.

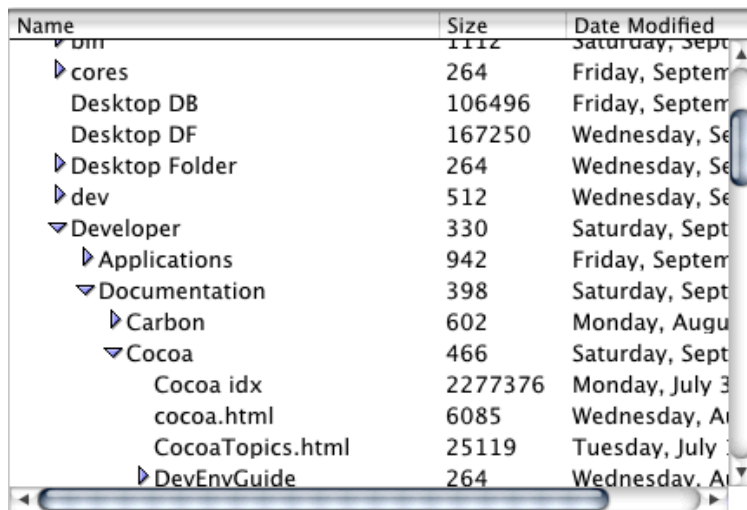
**Important:** `NSOutlineView` requires that each item in the outline view be unique.

An item in an outline view is expandable if it can contain other items. An expandable item is distinguished visually by a disclosure triangle, which points to the right when the item is collapsed and points down when the item is expanded. Clicking on the disclosure triangle causes the item to be expanded or collapsed, depending on the new state of the triangle. An item can be expanded even if it contains no items. An Option-click on an item's disclosure triangle expands or collapses all of its contained items.

When an item is expanded, the outline view can display the previous expanded or collapsed state of its contained items, if the items were previously shown. To automatically restore the entire expanded state of an outline view for previously shown items, use `setAutosaveExpandedItems:`.

Items inside an expanded item are indented. By default, as a user expands or collapses nested items, the width of the column is resized so that it is just wide enough to display the widest item, based on the width of the items and their indentation in the hierarchy. Justification follows the current system justification. To turn off automatic resizing, use `setAutoresizesOutlineColumn:`. Note that an item may consist of text, an image, or anything else that can be drawn by a subclass of `NSCell`.

An instance of `NSOutlineView` is typically displayed in an instance of `NSScrollView`, as shown below.



Name	Size	Date Modified
▶ bin	1112	Saturday, Sept
▶ cores	264	Friday, Septem
Desktop DB	106496	Friday, Septem
Desktop DF	167250	Wednesday, Se
▶ Desktop Folder	264	Wednesday, Se
▶ dev	512	Wednesday, Se
▼ Developer	330	Saturday, Sept
▶ Applications	942	Friday, Septem
▼ Documentation	398	Saturday, Sept
▶ Carbon	602	Monday, Augu
▼ Cocoa	466	Saturday, Sept
Cocoa idx	2277376	Monday, July 3
cocoa.html	6085	Wednesday, Au
CocoaTopics.html	25119	Tuesday, July
▶ DevEnvGuide	264	Wednesday, Au

## Behavior Inherited from NSTableView

An outline view inherits much of its behavior from its parent class, `NSTableView`. As a result, many operations supported by a table view, such as selecting rows or columns, repositioning columns by dragging column headers, deferred activation for dragging, and so on, are also supported by an outline view. Your application has control of these features, and can configure the view's parameters to allow or disallow certain operations. For example, you might choose not to allow editing or rearranging for specific columns.

The `NSTableView` class also provides methods for working with data, responding to mouse clicks, setting grid attributes, editing cells, and performing other operations. For more information, see *Table View Programming Guide*.

# Writing an Outline View Data Source

---

Outline views support a data-source delegate in addition to a standard delegate object. The data-source implements the `NSOutlineViewDelegate` protocol and provides data and information about that data to the outline view, and is responsible for managing the data. Although an outline view does not require a delegate, it must have a data source to display information.

## Data Source Requirements

Like an instance of `NSTableView`, an instance of `NSOutlineView` gets all of its data from an object that you provide, called its data source. Your data source object can store records in any way you choose, but it must be able to identify them by their position in the hierarchy through the `NSOutlineViewDataSource` protocol (prior to Mac OS X v10.6, this was an informal protocol—`NSOutlineViewDataSource`). The data source must minimally implement the data access methods (`outlineView:child:ofItem:`, `outlineView:isItemExpandable:`, `outlineView:numberOfChildrenOfItem:`, and `outlineView:objectValueForTableColumn:byItem:`). To specify the root item in any of these methods, `nil` is sent as the method's `item` argument. If you want to allow the user to edit items, you must also implement a method for changing the value of an attribute (`outlineView:setObject:forTableColumn:byItem:`).

Typically the data source itself manages a collection of model objects each of which knows what their value is, whether they represent a leaf node, and how many (if any) child objects they have.

## The Data Source and Memory Management

Just like a table view, an outline view uses the data source solely to get information. An outline view does not own its data source (see *Communicating With Objects*). Similarly, it does not own the objects it gets from the data source—if they are released your application is likely to crash unless you tell the outline view to reload its data.

The data source is a controller object, and you are responsible for ensuring that it is not deallocated before the outline view is finished with it (typically the data source is an object such as the document object in a document-based application, so there is no additional work to do). The data source is in turn responsible for retaining all of the objects it provides to an outline view, and updating the outline view when there's a change to the model. It is therefore not safe to release the root item—or any children—until you're no longer displaying it in the outline view. If you need to dispose of the root item, then you should ensure that references to it are nullified, and that the outline view is updated to ensure that no attempt is made to display other items that may also have been disposed of, as in the following example.

```
[rootItem release];
rootItem = nil;
[outlineView reloadData];
```

## Sample Data Source Implementation

The following example shows the implementation of a data source class used in conjunction with an outline view to display contents of the file system, and of a class used to represent entries in the file system. Listing 1 shows the implementation of the data source class. Listing 2 shows the implementation of a class used to represent entries in the file system. The singleton `rootItem` instance is used as the root object in the example in Listing 1.

### Listing 1 Implementation of Outline View Data Source

```
@implementation DataSource
// Data Source methods

- (NSInteger)outlineView:(NSOutlineView *)outlineView
numberOfChildrenOfItem:(id)item {

    return (item == nil) ? 1 : [item numberOfChildren];
}

- (BOOL)outlineView:(NSOutlineView *)outlineView isItemExpandable:(id)item {
    return (item == nil) ? YES : ([item numberOfChildren] != -1);
}

- (id)outlineView:(NSOutlineView *)outlineView child:(NSInteger)index
ofItem:(id)item {

    return (item == nil) ? [FileSystemItem rootItem] : [(FileSystemItem *)item
childAtIndex:index];
}

- (id)outlineView:(NSOutlineView *)outlineView
objectValueForTableColumn:(NSTableColumn *)tableColumn byItem:(id)item {
    return (item == nil) ? @"/" : [item relativePath];
}

@end
```

### Listing 2 Implementation of Outline View Data Source Item

```
@interface FileSystemItem : NSObject
{
    NSString *relativePath;
    FileSystemItem *parent;
    NSMutableArray *children;
}

+ (FileSystemItem *)rootItem;
- (NSInteger)numberOfChildren; // Returns -1 for leaf nodes
- (FileSystemItem *)childAtIndex:(NSUInteger)n; // Invalid to call on leaf nodes
- (NSString *)fullPath;
- (NSString *)relativePath;
```

```

@end

@implementation FileSystemItem

static FileSystemItem *rootItem = nil;
static NSMutableArray *leafNode = nil;

+ (void)initialize {
    if (self == [FileSystemItem class]) {
        leafNode = [[NSMutableArray alloc] init];
    }
}

- (id)initWithPath:(NSString *)path parent:(FileSystemItem *)parentItem {
    self = [super init];
    if (self) {
        relativePath = [[path lastPathComponent] copy];
        parent = parentItem;
    }
    return self;
}

+ (FileSystemItem *)rootItem {
    if (rootItem == nil) {
        rootItem = [[FileSystemItem alloc] initWithPath:@"/" parent:nil];
    }
    return rootItem;
}

// Creates, caches, and returns the array of children
// Loads children incrementally
- (NSArray *)children {

    if (children == nil) {
        NSFileManager *fileManager = [NSFileManager defaultManager];
        NSString *fullPath = [self fullPath];
        BOOL isDir, valid;

        valid = [fileManager fileExistsAtPath:fullPath isDirectory:&isDir];

        if (valid && isDir) {
            NSArray *array = [fileManager contentsOfDirectoryAtPath:fullPath
error:NULL];

            NSUInteger numChildren, i;

            numChildren = [array count];
            children = [[NSMutableArray alloc] initWithCapacity:numChildren];

            for (i = 0; i < numChildren; i++)
            {
                FileSystemItem *newChild = [[FileSystemItem alloc]
initWithPath:[array objectAtIndex:i]
parent:self];
                [children addObject:newChild];
            }
        }
    }
}

```

```

        [newChild release];
    }
}
else {
    children = leafNode;
}
}
return children;
}

- (NSString *)relativePath {
    return relativePath;
}

- (NSString *)fullPath {
    // If no parent, return our own relative path
    if (parent == nil) {
        return relativePath;
    }

    // recurse up the hierarchy, prepending each parent's path
    return [[parent fullPath] stringByAppendingPathComponent:relativePath];
}

- (FileSystemItem *)childAtIndex:(NSUInteger)n {
    return [[self children] objectAtIndex:n];
}

- (NSInteger)numberOfChildren {
    NSArray *tmp = [self children];
    return (tmp == leafNode) ? (-1) : [tmp count];
}

- (void)dealloc {
    if (children != leafNode) {
        [children release];
    }
    [relativePath release];
    [super dealloc];
}

@end

```

# Using an Outline View Delegate

---

The `NSOutlineViewDelegate` protocol gives the delegate control over the appearance of individual cells in the table, over changes in selection, and over editing of cells.

Delegate methods that request permission to alter the selection or edit a value are invoked during user actions that affect the outline view, but are not invoked by programmatic changes to the view. When making changes programmatically, you decide whether you want the delegate to intervene and, if so, you send the appropriate message (checking first that the delegate responds to that message). Because the delegate methods involve the actual data displayed by the outline view, the delegate is typically the same object as the data source, though this is not a requirement.

The `NSOutlineViewDelegate` protocol defines these delegate messages:

- `outlineView:willDisplayCell:forTableColumn:item:` informs the delegate that the outline view is about to draw the cell specified by the passed column and item. The delegate can modify the instance of `NSCell` provided to alter the display attributes for that cell; for example, making uneditable values display in italic or gray text.
- `outlineView:shouldSelectItem:` and `outlineView:shouldSelectTableColumn:` give the delegate control over whether the user can select a specified row or column (though the user can still reorder columns). This is useful for disabling a specified row or column. For example, in a database client application, when a user is editing a record you might want to not allow other users to select the same row.
- `selectionShouldChangeInOutlineView:` allows the delegate to deny a change in selection. For example, if the user is editing a cell and enters an improper value, the delegate can prevent the user from selecting or editing any other cells until a proper value has been entered into the original cell.
- `outlineView:shouldEditTableColumn:item:` asks the delegate whether it's okay to edit the cell specified by the passed column and item. The delegate can approve or deny the request.

The `NSOutlineViewDelegate` protocol defines these additional delegate messages:

- `outlineView:shouldExpandItem:` asks the delegate whether it's okay to expand the specified item.
- `outlineViewItemWillExpand:` informs the delegate that the outline view is about to expand the specified item.
- `outlineView:shouldCollapseItem:` asks the delegate whether it's okay to collapse the specified item.
- `outlineViewItemWillCollapse:` informs the delegate that the outline view is about to collapse the specified item.
- `outlineView:willDisplayOutlineCell:forTableColumn:item:` informs the delegate that the outline view is about to display the cell that includes the expansion symbol.

In addition to these methods, the delegate protocol is automatically registered to receive messages corresponding to `NSOutlineView` notifications. These inform the delegate when the selection changes or is about to change, when a column is moved or resized, and when an item is expanded or collapsed:

Delegate Message	Notification
<code>outlineViewColumnDidMove:</code>	<code>NSNotification</code> <code>NSNotification</code>
<code>outlineViewColumnDidResize:</code>	<code>NSNotification</code> <code>NSNotification</code>
<code>outlineViewSelectionDidChange:</code>	<code>NSNotification</code> <code>NSNotification</code>
<code>outlineViewSelectionIsChanging:</code>	<code>NSNotification</code> <code>NSNotification</code>
<code>outlineViewItemDidExpand:</code>	<code>NSNotification</code> <code>NSNotification</code>
<code>outlineViewItemDidCollapse:</code>	<code>NSNotification</code> <code>NSNotification</code>



# Document Revision History

---

This table describes the changes to *Outline View Programming Topics*.

Date	Notes
2010-03-24	Updated datasource example code. Added links to formal protocols for the datasource and delegate.
2009-10-19	Updated to note new formal protocol on Mac OS X v10.6.
2006-06-28	Clarified memory management for data source.
2006-05-23	Added information to the section "About Outline Views."
	Noted that Option-clicking an item's disclosure triangle expands or collapses all of its contained items, and that an outline view can remember the expanded or collapsed state of contained items that were previously shown.
	Noted that <code>NSOutlineView</code> requires that each item in the outline view be unique.
	Corrected minor typos.
2004-12-02	Changed the title of the article "Using an Outline View Data Source" to "Writing an Outline View Data Source."
2004-08-31	Enhanced description of data source requirements and implementation in <a href="#">"Writing an Outline View Data Source"</a> (page 11).
2002-11-12	Revision history was added to existing topic. It will be used to record changes to the content of the topic.

