Table View Programming Guide

Cocoa > User Experience

2006-04-04

Apple Inc.
© 1997, 2006 Apple Computer, 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, Cocoa, Mac, Mac OS, and Objective-C are trademarks of Apple Inc., registered in the United States and other countries.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. 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 Table Views Programming Guide 7

Organization of This Document 7

About Tables 9

The Parts of a Table 11

Using a Table Delegate 13

Using a Table Data Source 15

Using Drag and Drop in Tables 17

Configuring Your Table View 17
Beginning a Drag Operation 18
Validating a Drag Operation 19
Accepting a Drop 19
Customizing Drag Behavior 20
Background Drags 20

Document Revision History 21

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

Listings

Using Drag and Drop in Tables 17

Listing 1 Registering the table's supported data types 17
Listing 2 Initiating a drag from a table. 18

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

Introduction to Table Views Programming Guide

A table displays data for a set of related records, with rows representing individual records and columns representing the attributes of those records.

Organization of This Document

This document contains the following articles:

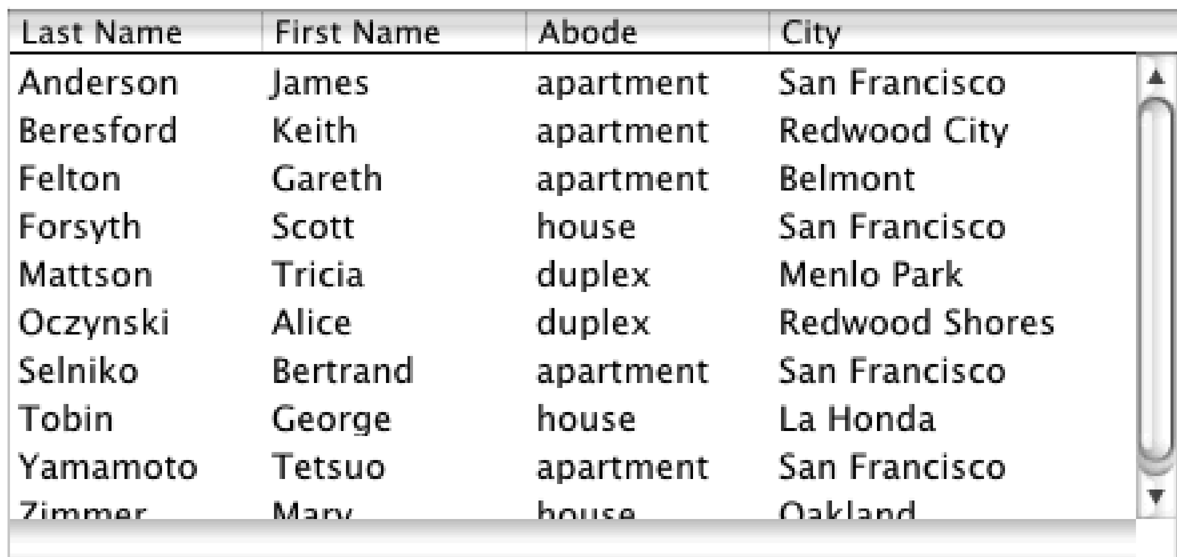
- "About Tables" (page 9) gives basic information on table views.
- "The Parts of a Table" (page 11) describes how all the different classes used by a table fit together.
- "Using a Table Data Source" (page 15) describes how to create a data source for your table.
- "Using a Table Delegate" (page 13) describes delegate methods for a table.
- "Using Drag and Drop in Tables" (page 17) describes how to support drag and drop in your table.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

About Tables

An `NSTableView` object displays data for a set of related records, with rows representing individual records and columns representing the attributes of those records. A record is a set of values for a particular real-world entity, such as an employee or a bank account. For example, in a table of employee records, each row represents one employee, and the columns represent such attributes as the first and last name, address, salary, and so on. A table view is usually displayed inside a scroll view, like this:



Last Name	First Name	Abode	City
Anderson	James	apartment	San Francisco
Beresford	Keith	apartment	Redwood City
Felton	Gareth	apartment	Belmont
Forsyth	Scott	house	San Francisco
Mattson	Tricia	duplex	Menlo Park
Oczynski	Alice	duplex	Redwood Shores
Selniko	Bertrand	apartment	San Francisco
Tobin	George	house	La Honda
Yamamoto	Tetsuo	apartment	San Francisco

In this illustration, the `NSTableView` object draws the table values while auxiliary views draw the column headers and adornments such as the blank square above the vertical scroller. The roles of these auxiliary views are discussed in "The Parts of a Table" (page 11).

The user selects rows or columns in the table by clicking them. The user edits individual cells by double-clicking. The user can rearrange columns by dragging the column headers and can resize the columns by dragging the divider between two column headers. You can configure the table to support the selection of multiple rows or columns. You can also configure it to support no selected rows or columns, which you might do to prevent the user from editing or rearranging them. You can also specify an action message to be sent when the user double-clicks something other than an editable cell.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

The Parts of a Table

A table is implemented by several classes, including NSTableView, NSTableHeaderView, NSTableColumn, NSTableHeaderCell, and possibly others. The primary class is NSTableView.

A table view is usually displayed together with a corner view and header view inside a scroll view (NSScrollView) object. The default corner view is a simple view object that fills in the corner above the vertical scroller. The header view is usually an instance of NSTableHeaderView that draws the column headers and handles column selection, rearranging, and resizing. If a table view is enclosed by a scroll view, the table view is the document view of the scroll view object.

You can replace the default corner view and header view of a table with custom views if you want. For example, you might replace the corner view with a button that sorts the table contents using the selected column. To replace the views, you use the setCornerView: and setHeaderView: methods of NSTableView. During window layout, the scroll view retrieves your custom views using the cornerView and headerView methods of NSTableView and tiles them along with the scrollers and document view.

Because the NSTableView and NSTableHeaderView objects both need access to column information (such as the column width), this information is encapsulated in NSTableColumn objects. An NSTableColumn object stores the width of the column, an attribute identifier used by the data source to identify the column contents, and flags that indicate whether the user can resize the column or edit its cells. (For information about the attribute identifier of a column, see "Using a Table Data Source" (page 15).) The object also stores two NSCell objects. The first cell object is used by the header view to draw the column header. The second cell object is used by the NSTableView to draw the data values in the column; the table column object reuses the same NSCell object for each row in the column.

The cell used to draw each column header is an instance of the NSTableHeaderCell class by default. An NSTableHeaderCell object contains the title displayed over the column as well as the font and color of that title. You use the API of its superclasses, NSTextFieldCell and NSCell, to set a column's title and to specify display attributes for that title (font, alignment, and so on). You can also use the setImage: method of NSCell to display an image instead of a title. To remove the image and restore the title, use the setStringValue: method of NSCell.

The cell used to draw the column's data values is typically an instance of NSTextFieldCell but can be an instance of any NSCell subclass, such as NSImageCell. This object is used to draw all values in the column and determines the font, alignment, text color, and other such display attributes for those values. You can customize the presentation of various kinds of values by assigning an NSFormatter object to the cell. For example, to properly display date values in a column, you could assign an NSDateFormatter object to the cell.

In Mac OS X version 10.3 and later, you can configure text in table cells to display an ellipsis glyph (…) when the column is too narrow to display the full text. To do this, you configure the table column cells with the message cell setWraps:YES. Then, for the object value returned by the data source, use an attributed string configured with the desired paragraph style. For a thorough explanation of this technique, see "Breaking Lines by Truncation".

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

Using a Table Delegate

In addition to the delegate methods used by other controls, a table view object has several table-specific delegate methods. These methods give 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 `NSTableView` 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, send the appropriate message (checking first that the delegate responds to that message).

Note: Because the delegate methods involve data displayed by the `NSTableView` object, the delegate is typically the same object as the data source. For more information about data source objects, see "Using a Table Data Source" (page 15).

The delegate can manage much of the table view behavior by implementing the following methods:

Delegate method	Description
`tableView: willDisplayCell: forTableColumn:row:`	Informs the delegate that the `NSTableView` object is about to draw a particular cell. The delegate can modify the provided `NSCell` object to alter the display attributes for that cell; for example, it could make uneditable values display in italic or gray text
`tableView: shouldSelectRow:` and `tableView: shouldSelectTableColumn:`	These methods give the delegate control over whether the user can select a particular row or column (though the user can still reorder columns). This is useful for disabling particular rows or columns. For example, in a database client application, when another user is editing a record you might want to prevent other users from selecting the row containing that record.
`selectionShould- ChangeInTableView:`	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.
`tableView: shouldEditTableColumn: row:`	Asks the delegate whether it is permitted to edit a particular cell. The delegate can approve or deny the request.

In addition to these methods, the delegate is also automatically registered to receive messages corresponding to `NSTableView` notifications. These messages inform the delegate when the selection changes and when a column is moved or resized:

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

Delegate Message	Notification
tableViewColumnDidMove:	NSTableViewColumnDidMoveNotification
tableViewColumnDidResize:	NSTableViewColumnDidResizeNotification
tableViewSelectionDidChange:	NSTableViewSelectionDidChangeNotification
tableViewSelectionIsChanging:	NSTableViewSelectionIsChangingNotification

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

Using a Table Data Source

Unlike most controls, an `NSTableView` object does not store or cache the data it displays. Instead, it gets all of its data from an object called a **data source** that you provide. Your data source object can store records in any way you choose, as long as it is able to identify those records by integer index. It must also implement methods to provide the following information:

- How many records are in the data source (`numberOfRowsInTableView` method)
- What is the value of a specific record (`tableView:objectValueForTableColumn:row:` method)

If your table supports the editing of records, you must also provide a `tableView:setObjectValue:forTableColumn:row:` method for changing the value of an attribute.

The `NSTableView` object treats objects provided by its data source as values to be displayed in `NSCell` objects. If these objects aren't of common value classes—such as `NSString`, `NSNumber`, and so on—you may need to create a custom `NSFormatter` object to display them. For more information, see *Data Formatting Programming Guide for Cocoa*.

The type of information stored in a particular field of a record is indicated by the identifier object of the corresponding column; see "The Parts of a Table" (page 11). Because columns can be reordered, you cannot rely on a column's position in the table to identify the type of data it contains. Instead, your data source must use the column's identifier object as a key to identifying the value. The identifier object can be any kind of object that uniquely identifies attributes for the data source. For example, you could use strings to identify the names of attributes such as "Last Name", "Address", and so on. The data source object could then use these strings as keys for `NSDictionary` objects.

Suppose that a table view's column identifiers are set up as strings containing the names of attributes for the column, such as "Last Name", "City", and so on. Also suppose that the data source stores each record in an `NSMutableDictionary` object and that all of the records are then stored in in an `NSMutableArray` object called *`records`*, Given this configuration, an ASCII property list version of the data might look like the following:

```
(
    {
        "Last Name" = Anderson;
        "First Name" = James;
        Abode = apartment;
        City = "San Francisco";
    },
    {
        "Last Name" = Beresford;
        "First Name" = Keith;
        Abode = apartment;
        City = "Redwood City";
    }
)
```


2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

With such a record structure, the following implementation of the `tableView:objectValueForTableColumn:row:` method is sufficient to retrieve values for the `NSTableView`:

```
- (id)tableView:(NSTableView *)aTableView
    objectValueForTableColumn:(NSTableColumn *)aTableColumn
    row:(int)rowIndex
{
    id theRecord, theValue;

    NSParameterAssert(rowIndex >= 0 && rowIndex < [records count]);
    theRecord = [records objectAtIndex:rowIndex];
    theValue = [theRecord objectForKey:[aTableColumn identifier]];
    return theValue;
}
```

The corresponding method for setting values would look like the following:

```
- (void)tableView:(NSTableView *)aTableView
    setObjectValue:anObject
    forTableColumn:(NSTableColumn *)aTableColumn
    row:(int)rowIndex
{
    id theRecord;

    NSParameterAssert(rowIndex >= 0 && rowIndex < [records count]);
    theRecord = [records objectAtIndex:rowIndex];
    [theRecord setObject:anObject forKey:[aTableColumn identifier]];
    return;
}
```

Finally, the `numberOfRowsInTableView:` method simply returns the count of the `NSArray` object:

```
- (int)numberOfRowsInTableView:(NSTableView *)aTableView
{
    return [records count];
}
```

In each case, the `NSTableView` object that sends the message is passed to the delegate method in the *`aTableView`* parameter. A data source object that manages several sets of data can choose the appropriate set based on which `NSTableView` object sent the message.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

Using Drag and Drop in Tables

The `NSTableView` class implements both the `NSDraggingSource` and `NSDraggingDestination` informal protocols. It manages most of the details of drag operations, sending messages to its data source when it begins or receives a drag operation. The minimum required steps for supporting drag and drop in a table are as follows:

1. Call the `registerForDraggedTypes:` method of your table view to specify the types of data your table supports.

2. Implement the `tableView:writeRowsWithIndexes:toPasteboard:` method in your data source to handle the beginning of a drag operation.

3. Implement the `tableView:validateDrop:proposedRow:proposedDropOperation:` method in your data source to validate the drop location.

4. Implement the `tableView:acceptDrop:row:dropOperation:` method in your data source to incorporate the dropped data.

Implementing the delegate methods in your data source object lets the `NSTableView` class know that your table supports drag and drop and also lets it know what to do with dragged or dropped data. If you omit one of these delegate methods from your data source, your table view will be unable to accept dropped data or initiate drags (depending on which method you omit). Other customizations to drag-and-drop behavior in table views are also possible but are not required.

The sections that follow provide additional information about how you implement the basic delegate methods and some specific customizations. For general information on how drag and drop works, see *Drag and Drop Programming Topics for Cocoa*.

Configuring Your Table View

The first step in configuring a table to support drag and drop is to tell it what data types your data source object understands. You do this by sending a `registerForDraggedTypes:` message to the `NSTableView` object. This method accepts an array of pasteboard types that your data source understands. If you want to support drag and drop operations only within your table, you can simply define a custom type. If you support data from several different sources, you might want to specify several different types.

The following example shows a sample implementation of the awakeFromNib method for a document. If you want to support drag and drop only within the table, you can simply register a custom type for your table data. If you wanted to accept other types of data, you would add the appropriate pasteboard types to the array.

Listing 1 Registering the table's supported data types

```
#define MyPrivateTableViewDataType @"MyPrivateTableViewDataType"
```


2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

```
- (void)awakeFromNib
{
    [myTableView registerForDraggedTypes:
                    [NSArray arrayWithObject:MyPrivateTableViewDataType] ];

    // Other initialization...
}
```

Beginning a Drag Operation

When a drag operation begins, the table sends a `tableView:writeRowsWithIndexes:toPasteboard:` message to the data source. Your implementation of this method should place the data for the specified rows onto the provided pasteboard and return `YES`. If, for some reason, you do not want the drag operation to continue, your method should return `NO`.

> **Note:** Prior to Mac OS X v10.4, table views initiated a drag by sending a `tableView:writeRows:toPasteboard:` message. In Mac OS X v10.4 and later, the use of this method is deprecated.

The following example shows a simple implementation of the `tableView:writeRowsWithIndexes:toPasteboard:` delegate method. This implementation assumes that drags and drops are confined to the table itself, so it simply copies the dragged row numbers to the pasteboard.

Listing 2 Initiating a drag from a table.

```
- (BOOL)tableView:(NSTableView *)tv writeRowsWithIndexes:(NSIndexSet *)rowIndexes
 toPasteboard:(NSPasteboard*)pboard
{
    // Copy the row numbers to the pasteboard.
    NSData *data = [NSKeyedArchiver archivedDataWithRootObject:rowIndexes];
    [pboard declareTypes:[NSArray arrayWithObject:MyPrivateTableViewDataType]
owner:self];
    [pboard setData:data forType:MyPrivateTableViewDataType];
    return YES;
}
```

If there is a lot of data to be placed onto the pasteboard, or if there are multiple data formats possible, you may provide the data lazily using promises. To do so, just tell the pasteboard object what types you support without providing the actual data. The pasteboard object will notify you at a later time if and when the data is actually needed. See the description of the `pasteboard:provideDataForType:` delegate method (Objective-C) or the `pasteboardProvideDataForType` delegate method (Java) in the `NSPasteboard` class for information about fulfilling a pasteboard promise.

In Mac OS X v10.4, support was added for handling file-promised drag operations in your data source object. To support this feature in a table view, you must first promise the the data to the pasteboard using the `NSFilesPromisePboardType` type in your `tableView:writeRowsWithIndexes:toPasteboard:` method. When a destination accepts the dropped file information, `NSTableView` calls through to the `tableView:namesOfPromisedFilesDroppedAtDestination:forDraggedRowsWithIndexes:` method of your data source to provide the files. Your implementation of this method should create the files and return an array containing the filenames (without path information).

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

The NSTableView class only supports local drags by default. When you try to drag table rows outside of the application, the table view's draggingSourceOperationMaskForLocal: method returns NSDragOperationNone. To allow interapplication drags in Mac OS X v10.4 and later, call the setDraggingSourceOperationMask:forLocal: method of NSTableView. (If your code supports versions of Mac OS X prior to v10.4, you must subclass NSTableView instead and override draggingSourceOperationMaskForLocal: to return an appropriate value.)

Validating a Drag Operation

When a drag operation enters a table view, the table view sends a tableView:validateDrop:proposedRow:proposedDropOperation: message to its data source. If this method is not implemented or if the method returns NSDragOperationNone, the drag operation is not allowed.

The last two parameters of the tableView:validateDrop:proposedRow:proposedDropOperation: method contain the proposed row insertion point and insertion behavior (NSTableViewDropOn or NSTableViewDropAbove). You can override these values in your delegate method implementation by sending a setDropRow:dropOperation: message to the table view.

Para

```
- (NSDragOperation)tableView:(NSTableView*)tv validateDrop:(id
<NSDraggingInfo>)info proposedRow:(int)row
proposedDropOperation:(NSTableViewDropOperation)op
{
    // Add code here to validate the drop
    NSLog(@"validate Drop");
    return NSDragOperationEvery;
}
```

Accepting a Drop

When a validated drag operation is dropped onto a table view, the table view sends a tableView:acceptDrop:row:dropOperation: message to its data source. The data source's implementation of this method should incorporate the data from the dragging pasteboard (obtained from the *acceptDrop* parameter) and use the other parameters to update the table. For example, if the drag operation type (also obtained from the *acceptDrop* parameter) was NSDragOperationMove and the drag originated from the table, you would want to move the row from its old location to the new one.

```
- (BOOL)tableView:(NSTableView *)aTableView acceptDrop:(id <NSDraggingInfo>)info
            row:(int)row dropOperation:(NSTableViewDropOperation)operation
{
    NSPasteboard* pboard = [info draggingPasteboard];
    NSData* rowData = [pboard dataForType:MyPrivateTableViewDataType];
   NSIndexSet* rowIndexes = [NSKeyedUnarchiver unarchiveObjectWithData:rowData];
    int dragRow = [rowIndexes firstIndex];

    // Move the specified row to its new location...
}
```


2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

Customizing Drag Behavior

Dragging the cursor vertically in a table view can be interpreted as an attempt either to select a range of rows or to drag one or more rows elsewhere. The default behavior of `NSTableView` interprets vertical drags as the beginning of a drag operation but you can change this behavior using the `setVerticalMotionCanBeginDrag:` method of `NSTableView`. Horizontal drags always begin a drag operation.

When a drag operation begins, the table view constructs an image to represent the dragged rows. The default image is a copy of the rows. If you want a different image, you must subclass `NSTableView` and override the `dragImageForRows:event:dragImageOffset:` method to return your own `NSImage` object.

Background Drags

Table views and outline views allow you to drag entries while your application is in the background. This only affects clients using the row/item based dragging APIs. The behavior when dragging from such a table is the same as the `NSTextView` implementation of dragging. If the user clicks and drags on your table while your application is not active, a drag operation is initiated. If the user simply clicks on your table, your application becomes active and the current selection does not change.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Table View Programming Guide*.

Date	Notes
2006-04-04	Updated drag and drop guidelines to mention registerForDraggedTypes: method.
	Title changed from *Table Views* to *Table Views Programming Guide*.
2004-08-31	Added information about truncating strings in cells to "The Parts of a Table" (page 11).
2002-11-12	Revision history added to existing topic.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 1997, 2006 Apple Computer, Inc. All Rights Reserved.