Accessibility Programming Guide for iOS

User Experience

2010-07-07

Apple Inc.
© 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, iPhone, iPod, Mac, Mac OS, and Safari are trademarks of Apple Inc., registered in the United States and other countries.

Numbers is a trademark of Apple Inc.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

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

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Figures and Listings

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Introduction

Using iOS 3.0 and later, VoiceOver is available to help users with visual impairments use their iOS–based devices. The UI Accessibility programming interface, introduced in iOS 3.0, helps developers make their applications accessible to VoiceOver users. Briefly, VoiceOver describes an application's user interface and helps users navigate through the application's views and controls, using speech and sound. Users familiar with VoiceOver in Mac OS X can leverage their experience to help them quickly come up to speed using VoiceOver on their devices.

iPhone applications that run in iOS 3.0 and later should be accessible to VoiceOver users. iOS and the iPhone SDK support this goal by:

- Making standard UIKit controls and views accessible by default
- Supplying the UI Accessibility programming interface, which defines a streamlined process for making an iPhone application accessible
- Providing tools that help you implement accessibility in your code and test the accessibility of your application

If you're developing or updating an iPhone application, you should read this document to learn how to make your application accessible to VoiceOver users.

Organization of This Document

This document contains the following chapters:

- "Accessibility on iPhone" (page 9) briefly describes how VoiceOver works on the device and introduces the programming interface and tools you can use to make your application accessible.
- "Making Your iPhone Application Accessible" (page 13) provides in-depth guidance for making your application accessible to VoiceOver users.
- "Testing the Accessibility of Your iPhone Application" (page 27) introduces the Accessibility Inspector, and describes how to use it and VoiceOver to test the accessibility of your application.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Accessibility on iPhone

From the beginning, iOS–based devices included several features that made the device easy for everyone to use, including visual voicemail, large fonts in Mail, and zooming in webpages, photos, and maps. With the addition of the following accessibility features, it's even easier for people with visual, auditory, and physical disabilities to use their devices:

- Zoom. Magnifies the entire device screen.
- White on Black. Inverts the colors on the display.
- Mono Audio. Combines the sound of the left and right channels into a mono signal played on both sides.
- Speak Auto-text. Speaks the text corrections and suggestions iPhone makes while users type.
- Voice Control. Allows users to make phone calls and control iPod playback using voice commands.

In addition, visually impaired users can rely on VoiceOver to help them use their devices.

iPhone Accessibility and VoiceOver

VoiceOver is Apple's innovative screen-reading technology, which gives users control over their devices without having to see the screen. VoiceOver does this by acting as an intermediary between an application's user interface and the user's touch, providing audible descriptions of elements and actions in the application. When VoiceOver is active, users don't have to worry about accidentally deleting a contact or calling a phone number, because VoiceOver tells them where they are in the user interface, what actions they can take, and what the results of those actions will be.

An application is accessible when all user interface elements with which users can interact are accessible. A user interface element is accessible when it properly reports itself as an accessibility element.

To be useful, however, an accessible user interface element must provide accurate and helpful information about its screen position, name, behavior, value, and type. This is the information VoiceOver speaks to users. The iPhone SDK contains a programming interface and tools that help you ensure that the user interface elements in your application are both accessible and useful (for more information, see "iPhone Accessibility API and Tools" (page 10)).

Why Should You Make Your Application Accessible?

You should make your iPhone application accessible to VoiceOver users because:

- It increases your user base. You've worked hard to create a great application; don't miss the opportunity to make it available to even more users.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

- It allows people to use your application without seeing the screen. Users with visual impairments can use your application with the help of VoiceOver.
- It helps you address accessibility guidelines. Various governing bodies create guidelines for accessibility and making your iPhone application accessible to VoiceOver users can help you meet them.
- It's the right thing to do.

It's important to be aware that supporting accessibility does not impact your ability to innovate and create beautiful iPhone applications. The UI Accessibility programming interface allows you to add a thin layer of functionality that does not alter your application's appearance, or interfere with its main logic.

iPhone Accessibility API and Tools

iOS 3.0 and later includes the UI Accessibility programming interface, which is a lightweight API that helps an application provide all the information VoiceOver needs to describe the user interface and help visually impaired people use the application.

The UI Accessibility programming interface is part of UIKit and is implemented on standard UIKit controls and views by default. This means that, when you use standard controls and views, much of the work of making your application accessible is done for you. Depending on the level of customization in your application, making it accessible can be as simple as providing accurate and helpful descriptions of your accessible user-interface elements.

The iPhone SDK also provides tools to help you make your application accessible:

- An Interface Builder inspector pane that provides an easy way to furnish descriptive accessibility information while you're designing your nib files. To learn more about how to do this, see "Defining Custom Attribute Information in Interface Builder" (page 20).
- Accessibility Inspector, which displays the accessibility information embedded in your application's user interface and allows you to verify this information when you run your application in iPhone Simulator. To learn how to examine the accessibility information in your application, see "Using Accessibility Inspector to Test Your Application" (page 27).

In addition, you can use VoiceOver itself to test the accessibility of your application. To learn how to test your application with VoiceOver, see "Using VoiceOver to Test Your Application" (page 30).

The UI Accessibility Programming Interface

The UI Accessibility programming interface consists of two informal protocols, one class, a function, and a handful of constants:

- The `UIAccessibility` informal protocol. Objects that implement the `UIAccessibility` protocol report their accessibility status (that is, whether they are accessible) and supply descriptive information about themselves. Standard UIKit controls and views implement the `UIAccessibility` protocol by default.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

- The `UIAccessibilityContainer` informal protocol. This protocol allows a subclass of `UIView` to make some or all of the objects it contains accessible as separate elements. This is particularly useful when the objects contained in such a view are not themselves subclasses of `UIView` and, for this reason, are not automatically accessible.
- The `UIAccessibilityElement` class. This class defines an object that can be returned through the `UIAccessibilityContainer` protocol. You can create an instance of `UIAccessibilityElement` to represent an item that isn't automatically accessible, such as an object that does not inherit from `UIView`, or an object that does not exist.
- The `UIAccessibilityConstants.h` header file. This header file defines the constants that describe the traits that an accessibility element can exhibit and the notifications that an application can post.

Accessibility Attributes

The attributes that describe an accessible user interface element make up the core of the UI Accessibility API. VoiceOver supplies attribute information to users when they access or interact with a control or view.

Attributes are also the components of the programming interface that you're most likely to use. This is because attributes encapsulate the information that differentiates one control or view from another. For standard UIKit controls and views, you might just need to ensure that the default attribute information is appropriate for your application; for custom controls and views, you might have to supply most of the attribute information.

The UI Accessibility programming interface defines the following attributes:

- **Label**. A short, localized word or phrase that succinctly describes the control or view, but does not identify the element's type. Examples are "Add" or "Play."
- **Traits**. A combination of one or more individual traits, each of which describes a single aspect of an element's state, behavior, or usage. For example, an element that behaves like a keyboard key and that is currently selected can be characterized by the combination of the Keyboard Key and Selected traits.
- **Hint**. A brief, localized phrase that describes the results of an action on an element. Examples are "Adds a title" or "Opens the shopping list."
- **Frame**. The frame of the element in screen coordinates, which is given by the `CGRect` structure that specifies an element's screen location and size.
- **Value**. The current value of an element, when the value is not represented by the label. For example, the label for a slider might be "Speed," but its current value might be "50%."

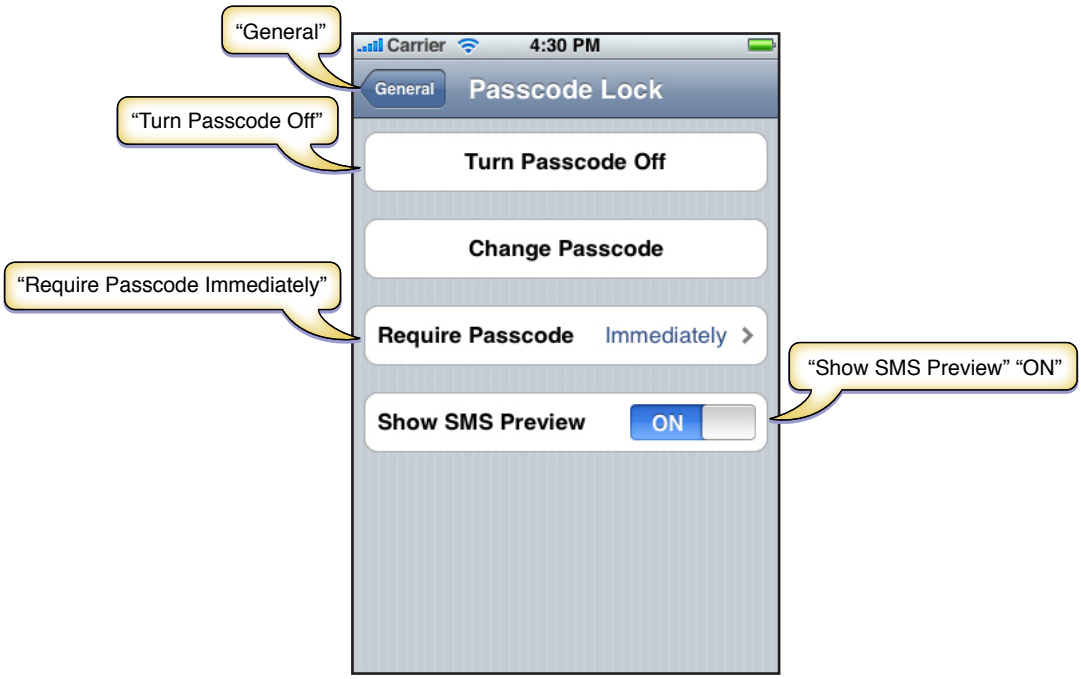
Accessibility elements provide content for attributes, whether that content is supplied by default or by you. An accessibility element always provides content for the frame and label attributes. The frame attribute is required because an accessibility element must always be able to report its position in the user interface. (Note that an object that inherits from `UIView` includes the frame attribute by default.) The label attribute is required because it contains the name or description of the accessibility element that VoiceOver speaks.

An accessibility element is not required to provide content for the hint and traits attributes, if these attributes do not apply to the element. For example, an element that does not perform an action does not need to provide a hint.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

An accessibility element provides information for the value attribute only when the element's contents are changeable and cannot always be described by the label. For example, a text field that contains an email address might have the label "Email address," but its contents depends on user input and is usually of the form "username@address." Figure 1-1 shows some of the information VoiceOver might provide.

Figure 1-1 VoiceOver speaks the information provided by accessible elements



VoiceOver users rely on the labels and hints they hear to help them use your application. For this reason, it's especially important to make sure that every accessible user interface element in your application provides accurate and informative descriptions. For standard UIKit controls and views, the default attribute information is often appropriate, but you should inspect these elements to make sure. For custom controls and views, you may have to supply some or all of this information yourself. For some guidelines on how to do this, see "Crafting Useful Labels and Hints" (page 16).

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Making Your iPhone Application Accessible

To be accessible, an iPhone application must supply information about its user interface elements to VoiceOver users. At a high level, this means you should make sure that:

- Every user interface element with which users can interact is accessible. This includes elements that merely supply information, such as static text, as well as controls that perform actions.
- All accessible elements supply accurate and helpful information.

In addition to these fundamentals, there are also a few things you can do to enhance a VoiceOver user's experience with table views and to make sure dynamic elements in your application are always accessible.

Make User Interface Elements Accessible

As mentioned in "iPhone Accessibility and VoiceOver" (page 9) a user interface element is accessible if it reports itself as an accessibility element. Although being accessible is not enough to make a user interface element useful to VoiceOver users, it represents the first step in the process of making your application accessible.

As stated in "iPhone Accessibility API and Tools" (page 10), standard UIKit controls and views are automatically accessible. If you use only standard UIKit controls, you probably don't have to do much additional work to make sure your application is accessible. In this case, your next step is to ensure that the default attribute information supplied by these controls makes sense in your application. To learn how to do this, see "Supply Accurate and Helpful Attribute Information" (page 16).

If you create custom views that display information or with which users need to interact, you must ensure the accessibility of these views yourself. After you've done this, you need to make sure these views supply accessibility information that helps people use them (see "Supply Accurate and Helpful Attribute Information" (page 16)).

From the perspective of accessibility, a custom view is either an individual view or a container view. An **individual view** does not contain any other views that need to be accessible. For example, a custom subclass of `UIControl` that displays an icon and behaves like a button does not contain any other elements with which the user can interact, apart from the button itself. Read "Make Custom Individual Views Accessible" (page 14) to learn how to make an individual view accessible.

A **container view**, on the other hand, contains other elements with which users can interact. For example, in a custom subclass of `UIView` that performs its own drawing of geometric shapes, the shapes represent elements with which users can interact, and which are distinct from the container view. Such separate elements within a container view are not automatically accessible (because they are not subclasses of `UIView`) and do not provide any accessibility information. To learn how to make the contents of a container view accessible, read "Make the Contents of Custom Container Views Accessible" (page 14).

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Make Custom Individual Views Accessible

If your application contains a custom individual view with which users need to interact, you must make the view accessible. (Recall that an individual view is a view containing no other views with which users interact.)

In addition to using Interface Builder to make a custom individual view accessible, there are two programmatic ways to do this. One way is to set your custom view's accessibility status in the code that instantiates it. The following code snippet shows how to do this:

```
@implementation MyCustomViewController
- (id)init
{
  _view = [[[MyCustomView alloc] initWithFrame:CGRectZero] autorelease];
  [_view setIsAccessibilityElement:YES];

  /* Set attributes here. */
}
```

Another way is to implement the `isAccessibilityElement` method of the `UIAccessibility` protocol in the implementation of your custom subclass. The following code snippet shows how to do this:

```
@implementation MyCustomView
  /* Implement attribute methods here. */

- (BOOL)isAccessibilityElement
{
   return YES;
}
```

Note: In both of these code snippets, you can see comments in place of the code needed to set accessibility attributes. For more complete code snippets that show how to do this, see "Defining Custom Attribute Information Programmatically" (page 21).

Make the Contents of Custom Container Views Accessible

If your application displays a custom view that contains other elements with which users interact, you need to make the contained elements separately accessible. At the same time, you need to make sure that the container view itself is not accessible. The reason is that users interact with the contents of the container, not with the container itself.

To accomplish this, your custom container view should implement the `UIAccessibilityContainer` protocol. This protocol defines methods that make the contained elements available in an array.

The following code snippet shows the partial implementation of a custom container view. Note that this container view creates the array of accessible elements only when methods of the `UIAccessibilityContainer` protocol are called. As a result, if iPhone accessibility is not currently active, the array is not created.

Listing 2-1 Make the contents of a custom container view accessible as separate accessibility elements

```
@implementation MultiFacetedView
- (NSArray *)accessibleElements
{
```


2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

```
    if ( _accessibleElements != nil )
    {
        return _accessibleElements;
    }
    _accessibleElements = [[NSMutableArray alloc] init];

    /* Create an accessibility element to represent the first contained element
 and initialize it as a component of MultiFacetedView. */

    UIAccessibilityElement *element1 = [[[UIAccessibilityElement alloc]
initWithAccessibilityContainer:self] autorelease];

    /* Set attributes of the first contained element here. */

    [_accessibleElements addObject:element1];

    /* Perform similar steps for the second contained element. */

    UIAccessibilityElement *element2 = [[[UIAccessibilityElement alloc]
initWithAccessibilityContainer:self] autorelease];

    /* Set attributes of the second contained element here. */

    [_accessibleElements addObject:element2];
}

/* The container itself is not accessible, so MultiFacetedView should return NO
 in isAccessiblityElement. */

- (BOOL)isAccessibilityElement
{
    return NO;
}

/* The following methods are implementations of UIAccessibilityContainer protocol
 methods. */

- (NSInteger)accessibilityElementCount
{
    return [[self accessibleElements] count];
}

- (id)accessibilityElementAtIndex:(NSInteger)index
{
    return [[self accessibleElements] objectAtIndex:index];
}

- (NSInteger)indexOfAccessibilityElement:(id)element
{
    return [[self accessibleElements] indexOfObject:element];
}
@end
```


2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Supply Accurate and Helpful Attribute Information

There are two parts to the process of supplying attribute information for accessible elements:

- Crafting concise, accurate, and helpful information
- Ensuring that the accessible elements in your application report that content correctly

If you use custom views, you must supply all appropriate attribute information for them. For guidance, see "Guidelines for Creating Useful Labels" (page 16), "Guidelines for Creating Useful Hints" (page 17), and "Identifying Appropriate Traits" (page 18).

Even if you use only standard UIKit controls and views, you might find that some of the default attribute information they supply could be enhanced to make more sense in the context of your application. For more information, see "Enhancing Default Attribute Information" (page 16).

If you need to supply or change accessibility attributes on either standard or custom UI elements, you can do so either in Interface Builder (see "Defining Custom Attribute Information in Interface Builder" (page 20)) or programmatically (see "Defining Custom Attribute Information Programmatically" (page 21)).

Enhancing Default Attribute Information

As part of the built-in accessibility of standard UIKit controls and views, iOS also provides default attribute information that describes these elements to VoiceOver users. In most cases, this information is appropriate for applications that use the standard controls and views. However, there might be times when supplying custom attribute information can enhance a VoiceOver user's experience with your application:

- If you use a standard UIKit control or view that displays a system-provided icon or title, first make sure you're using it in accordance with its intended purpose (see *iPhone Human Interface Guidelines* for more information). Then, decide whether the default label attribute accurately conveys the result of using that control or view in your application. If not, consider providing a hint attribute.

  For example, if you place an Add button in your navigation bar by using the system-provided Add (+) icon in a `UIBarButtonItem` object, you get the default label attribute, Add, automatically. If it's always obvious which item the user is adding when they activate this button, there's no need to provide a hint attribute. But if there might be confusion, you should consider providing a custom hint that describes the result of using that control in your application, such as "Adds an account" or "Adds a comment."
- If you display a custom icon or image in a standard UIKit view, such as a `UIButton` object, you need to supply a custom label attribute that describes it.

Crafting Useful Labels and Hints

When VoiceOver users run your application, they rely on the descriptions VoiceOver speaks to understand what your application does and how to use it. Because these descriptions represent the bulk of the VoiceOver user's experience with your application it's essential that they be as accurate and helpful as possible. The guidelines in this section help you create labels and hints that will make your application easy and enjoyable for people with disabilities to use.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Guidelines for Creating Labels

The label attribute identifies the user interface element. Every accessible user interface element, standard and custom, must supply content for the label attribute.

Note: A table row can also have a label attribute. However, the guidelines for creating a table row's label differ from the guidelines for creating labels for other types of controls and views. See "Enhance the Accessibility of Table Views" (page 22) to learn how to create useful labels for table rows.

A good way to determine what a label should convey is to think about what a sighted user infers about your application just by looking at it. If you've designed a good user interface, sighted users should know what a control or view does in the current application context by reading its title or understanding its icon. This is the information you need to make available to VoiceOver users in the label attribute.

If you provide a custom control or view, or if you display a custom icon in a standard control or view, you need to provide a label that:

- **Very briefly describes the element**. Ideally, the label consists of a single word, such as Add, Play, Delete, Search, Favorites, or Volume.

  Strive to design your application so that a single word identifies an element and makes its usage obvious in the current context. Sometimes, however, it might be necessary to use a brief phrase to properly identify an element. When this is the case, create a very short phrase, such as "Play music," "Add name," or "Add to event."

- **Does not include the type of the control or view**. The type information is contained in the traits attribute of the element and should never be repeated in the label.

  For example, if you include the control type in the label of an Add button, VoiceOver users hear "Add button button" every time they access that control. This experience would quickly become annoying and might motivate users to stop using your application.

- **Begins with a capitalized word**. This helps VoiceOver read the label with the appropriate inflection.

- **Does not end with a period**. The label is not a sentence and therefore should not end with a period.

- **Is localized**. Be sure to make your application available to as wide an audience as possible by localizing all strings, including accessibility attribute strings. In general, VoiceOver speaks in the language that the user specifies in International settings.

Guidelines for Creating Hints

The hint attribute describes the results of performing an action on a control or view. You should provide a hint only when the results of an action are not obvious from the element's label.

For example, if you provide a Play button in your application, the context in which the button appears should make it easy for users to understand what happens when they tap it. However, if you allow users to play a song by tapping the song title in a list, you might want to provide a hint that describes this result. The reason is that the label of the list item describes the item itself (in this case, the song title), not what happens when a user taps it.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Note: VoiceOver users can choose whether to hear available hints by selecting an option in VoiceOver settings on their device. Spoken hints are turned on by default.

If the results of a user's action on a control or view are not clearly implied by its label, create a hint that:

- **Very briefly describes the results**. Even though few controls and views need hints, strive to make the hints you do need to provide as brief as possible. Doing so decreases the amount of time users must spend listening before they can use the element.

  That said, however, avoid sacrificing clarity and good grammar for brevity. For example, changing "Adds a city" to "Adds city" does not significantly decrease the length of the hint, but does make it sound awkward and a bit less clear.

- **Begins with a plural verb**. As you create a hint, imagine that you're describing the use of a control to a friend. You might say something like "Tapping this control plays the song." Often, you can use the second phrase in such a sentence (in this case, "Plays the song") as a hint.

  Avoid beginning the phrase with a singular verb, because this can make the hint sound like a command. In other words, you don't want to tell users to "Play the song"; instead, you want to tell users that tapping the element "Plays the song."

- **Begins with a capitalized word and ends with a period**. Even though a hint is a phrase, not a sentence, ending the hint with a period helps VoiceOver speak it with the appropriate inflection.

- **Does not include the name of the action or gesture**. A hint does not tell users how to perform the action, it tells users what will happen when that action occurs. Therefore, do not create hints such as "Tap to play the song," "Tapping purchases the item," or "Swipe to delete the item."

  This is especially important because VoiceOver users can use VoiceOver-specific gestures to interact with elements in your application. If you name a different gesture in a hint, it would be very confusing.

- **Does not include the name of the control or view**. The user gets this information from the label attribute, so you should not repeat it in the hint. Therefore, do not create hints such as "Save saves your edits" or "Back returns to the previous screen."

- **Does not include the type of the control or view**. The user already knows whether, for example, the control or view behaves like a button or a search field, because this information is available in the element's traits attribute. Therefore, do not create hints such as "Button that adds a name" or "Slider that controls the scale."

- **Is localized**. As with accessibility labels, hints should be available in the user's preferred language.

Identifying Appropriate Traits

The traits attribute contains one or more individual traits that, taken together, describe the behavior of an accessible user interface element. Because some individual traits can be combined to describe a single element, the element's behavior can be precisely characterized.

Note: Individual traits are combined using the OR operator. Outside of code examples, this document uses the shorter terms "combine" and "combination" without specifying the method of combination.

A standard UIKit control, such as a button or text field, provides default content in the traits attribute. If you use only standard UIKit controls in your application (and do not customize their behavior in any way), you do not have to make any changes to the traits attribute of these controls.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

If you customize the behavior of a standard control, you might need to combine a new trait with the control's default traits. If you create a custom control or view, you need to provide content for the element's traits attribute.

The UI Accessibility programming interface defines 12 individual traits, some of which can be combined. Some traits characterize an element by identifying its behavior with the behavior of a particular type of control (such as a button) or type of object (such as an image). Other traits characterize an element by describing a specific behavior the element can exhibit, such the ability to play sound.

You use the following traits to characterize elements in your application:

- Button
- Link
- Search Field
- Keyboard Key
- Static Text
- Image
- Plays Sound
- Selected
- Summary Element
- Updates Frequently
- Not Enabled
- None

In general, traits that correspond to controls can successfully be combined with traits that describe behaviors. For example, you might combine the Button trait with the Plays Sound trait to characterize a custom control that behaves like a button and plays sound when it is tapped.

For the most part, you should consider the traits that correspond to particular controls, specifically, the Button, Link, Search Field, and Keyboard Key traits, to be mutually exclusive. That is, you should not use more than one of these traits to characterize an element in your application. Instead, think about which one of these four traits corresponds most closely to the element in your application. Then, if your element has additional behaviors, you can combine the first trait with one of the behavioral traits.

For example, suppose you display an image in your application that responds to a user's tap by opening a link in Safari on iPhone. You could characterize this element by combining the Image and Link traits. Another example is a keyboard key that modifies other keyboard keys when it is tapped. You could characterize this element by combining the Keyboard Key and Selected traits.

To see some examples of how traits characterize controls, you can use Accessibility Inspector to see the default traits that are set on the standard controls. For information on how to use Accessibility Inspector, read "Using Accessibility Inspector to Test Your Application" (page 27).

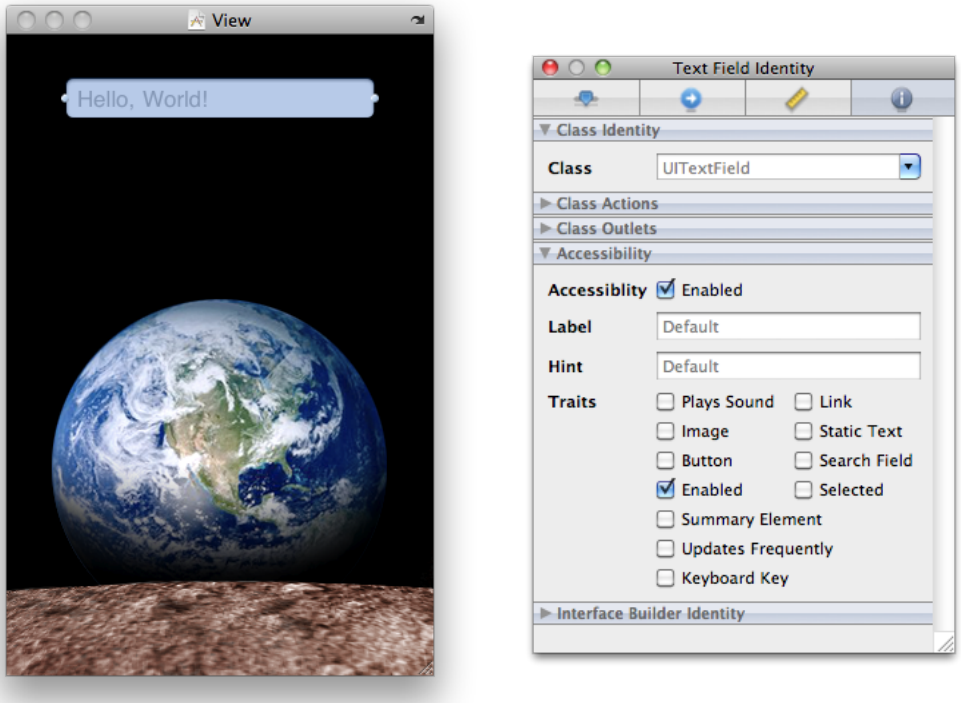
2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Defining Custom Attribute Information in Interface Builder

When you install iPhone SDK 3.0, you get a version of Interface Builder that includes features that help you make your application accessible. If your application contains standard UIKit controls and views, you might be able to do all your accessibility work in Interface Builder.

Using Interface Builder, you can set an element's accessibility status and provide custom content for the label, hint, and traits attributes. To do this, select the user interface element in your nib file and open the Identity inspector. Reveal the Accessibility section in the inspector and you should see something like Figure 2-1:

Figure 2-1 Default accessibility information for a standard text field, displayed in Interface Builder



As you can see in Figure 2-1, the standard text field used in the nib file is accessible by default and includes default information for the label, hint, and traits attributes. (Note that, for a text field that displays placeholder text, the default label is the placeholder text.) You can change any of these default values by supplying new information in the Identify inspector, as shown in Figure 2-2. (Figure 2-2 also shows how Accessibility Inspector displays the accessibility information for the text field. See "Using Accessibility Inspector to Test Your Application" (page 27) to learn about Accessibility Inspector.)

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Figure 2-2 Supplying accessibility information in Interface Builder

Defining Custom Attribute Information Programmatically

If you prefer, you can supply custom information for attributes programmatically. You might want to do this if you're not using Interface Builder at all or if you generate your views in code instead of using Interface Builder.

As described in "Make Custom Individual Views Accessible" (page 14), you can set accessibility information in the implementation of your view subclass or in the code that instantiates the view. Both techniques are valid, but there is one reason why you might want to implement attribute methods in your subclass instead of setting attributes in the instantiation code: If your view displays data that is dynamic or that changes frequently, such as the time of day, you should implement the subclass methods to return fresh data as needed. For those situations, if you only set attributes when you instantiate the subclass, the returned data is likely to be out of date.

The code snippets in this section build on those in "Make Custom Individual Views Accessible" (page 14), by including some of the attribute-specific methods. For example, if you want to implement accessibility methods in your subclass, you would write code similar to that in Listing 2-2.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Listing 2-2 Providing attribute information in a custom subclass implementation

```
@implementation MyCustomView
- (BOOL)isAccessibilityElement
{
   return YES;
}

- (NSString *)accessibilityLabel
{
   return NSLocalizedString(@"MyCustomView.label", nil);
}

/* This custom view behaves like a button. */
- (UIAccessibilityTraits)accessibilityTraits
{
   return UIAccessibilityTraitButton;
}

- (NSString *)accessibilityHint
{
   return NSLocalizedString(@"MyCustomView.hint", nil);
}
@end
```

If you want to use the property-setting methods of the `UIAccessibility` protocol in the code that instantiates your custom view, you can write code similar to that in Listing 2-3.

Listing 2-3 Providing attribute information in the code that instantiates a custom subclass object

```
@implementation MyCustomViewController
- (id)init
{
  _view = [[[MyCustomView alloc] initWithFrame:CGRectZero] autorelease];
  [_view setIsAccessibilityElement:YES];

  [_view setAccessibilityTraits:UIAccessibilityTraitButton];
  [_view setAccessibilityLabel:NSLocalizedString(@"view.label", nil)];
  [_view setAccessibilityHint:NSLocalizedString(@"view.hint", nil)];
}
```

Enhance the Accessibility of Table Views

If your application displays a table view in which each cell contains items other than (or in addition to) text, there are a few things you can do to make it more accessible. Similarly, if your table view displays more than one piece of information per row, you can enhance a VoiceOver user's experience by aggregating the information in a single, easy-to-understand label.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Note: If your table cells contain any of the standard table-view elements, such as the disclosure indicator, detail disclosure button, or delete control, you do not have to do anything to make these elements accessible. If, however, your table cells include other types of controls, such as a switch or a slider, you need to make sure that these elements are appropriately accessible.

If the table cells in your application contain a mix of different elements, determine whether users interact with each cell as a unit, or with individual elements inside the cell. If users need to access individual elements inside the cell, you should:

- Make each individual element accessible separately.
- Make sure the table cell itself is *not* accessible.
- Succinctly describe the overall contents of the cell and use this description for the label attribute of the cell. Note that, in this case, the label is considered to be one of the accessible elements within the cell.

You've probably recognized that a table cell that contains multiple items, such as text and controls, fits the criteria of a container view, as defined by the UI Accessibility programming interface. However, you do not have to identify the cell as a container view or implement any of the methods of the `UIAccessibilityContainer` protocol, because the table cell is automatically designated as a container.

If your table contains cells that provide information in discrete chunks, you should consider combining the information from these chunks in the label attribute. When you do this, VoiceOver users can get the meaning of the cell's contents with one gesture, instead of having to access each piece of information separately.

A good example of how this can work is in the built-in Stocks application. Instead of providing the company name, current stock price, and change in price as separate strings, Stocks combines this information in the label, which might sound like this: "Apple Inc., $125.25, up 1.3%." Notice the commas in this label. When you combine discrete pieces of information in this way, you can use commas to tell VoiceOver to pause briefly between phrases, making it easier for users to understand the information.

Here's a code snippet that shows how to combine the information in the labels of two separate elements into a single label that describes both:

```
@implementation CurrentWeather
/* This is a view that provides weather information. It contains a city subview
 and a temperature subview, each of which provides a separate label. */
- (NSString *)accessibilityLabel
{
   NSString *weatherCityString = [_weatherCity accessibilityLabel];
   NSString *weatherTempString = [_weatherTemp accessibilityLabel];

   /* Combine the city and temperature information so that VoiceOver users can
 get the weather information with one gesture. */

   return [NSString stringWithFormat:@"%@, %@", weatherCityString,
weatherTempString];
}
@end
```


2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Make Dynamic Elements Accessible

If user interface elements in your application can change dynamically, you need to make sure that the accessibility information they supply is accurate and up-to-date. You also need to send notifications when changes occur in the layout of application screens, so that VoiceOver can help users navigate the new layout. The UI Accessibility programming interface provides two notification types you can use when these kinds of changes occur on the screen. (To learn about these notifications, see "Notifications" in *UIAccessibility Protocol Reference*.)

If user interface elements can be in different states depending on other conditions in your application, you need to add logic to your code that returns the correct accessibility information for each state an element can be in. Because these changes can occur as the result of user actions, it's best to add this logic to a subclass's implementation, not to the code that instantiates the subclass.

The following code shows how to handle dynamic state changes and how to send a notification when a screen layout changes. The code represents the implementation of a `UIView` subclass that behaves like a custom keyboard key. The key's accessibility label changes depending on whether the instance represents a letter or other type of character, and on whether a shift key is currently selected. The key also returns different accessibility traits, depending on what type of keyboard key it represents and whether it is currently selected. Note that the code in Listing 2-4 assumes that there are a number of methods that query the state of the keyboard.

Listing 2-4 Returning the correct accessibility information for the current conditions and sending a layout-change notification

```
@implementation BigKey
- (NSString *)accessibilityLabel
{
   NSString *keyLabel = [_keyLabel accessibilityLabel];
   if ( [self isLetterKey] )
   {
      if ( [self isShifted] )
      {
         return [keyLabel uppercaseString];
      }
      else
      {
         return [keyLabel lowercaseString];
      }
   }
   else
   {
      return keyLabel;
   }
}

- (UIAccessibilityTraits)accessibilityTraits
{
   UIAccessibilityTraits traits = [super accessibilityTraits] |
UIAccessibilityTraitKeyboardKey;

   /* If this is the shift key and it's selected, users need to know that shift
 is currently in effect. */
   if ( [self isShiftKey] && [self isSelected] )
   {
```


2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

```
        traits |= UIAccessibilityTraitSelected;
    }

    return traits;
}

- (void)keyboardChangedToNumbers
{
    /* Code to perform the change to a number keyboard here. */

    /* Send a notification of this change to the screen layout. */

    UIAccessibilityPostNotification(UIAccessibilityNotificationLayoutChange,
nil);
}
@end
```

Make Nontextual Data Accessible

Sometimes your application displays data that's not automatically compatible with the way accessibility works. For example, if you display an image, you should provide a description of it in its accessibility label so that VoiceOver users can understand the information the image conveys. Or, if you provide information in a graphical way, such as a rating system that displays stars, you should make sure the accessibility label conveys the meaning behind the graphical representation.

The following code snippet uses the example of a custom view that draws the number of stars that corresponds to the rating of an item. The code shows how this view returns an appropriate accessibility label, depending on how many stars it draws.

```
@implementation RatingView
/* Other subclass implementation code here. */

- (NSString *)accessibilityLabel
{
    /* _starCount is an instance variable that contains how many stars to draw.
 */
    NSInteger starCount = _starCount;
    if ( starCount == 1 )
    {
        ratingString = NSLocalizedString(@"rating.singular.label", nil); // here,
 ratingString is "star"
    }
    else
    {
        ratingString = NSLocalizedString(@"rating.plural.label", nil); // here,
ratingString is "stars"
    }

    return [NSString stringWithFormat:@"%d %@", starCount, ratingString];
}
@end
```


2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Testing the Accessibility of Your iPhone Application

In iOS 3.0 and later, there are two complementary ways to test the accessibility of your application:

- The Accessibility Inspector in iPhone Simulator
- VoiceOver on the device

For the best results, you should test your application using both methods.

Using Accessibility Inspector to Test Your Application

Accessibility Inspector displays accessibility information about each accessible element in an application. You can use Accessibility Inspector to simulate VoiceOver interaction with the accessible elements in your application and examine the information they provide.

Note: Accessibility Inspector is ideal for testing the accessibility of your application during development, but it is no substitute for testing your application with VoiceOver. For one thing, Accessibility Inspector does not speak accessibility information, so you can't hear how your element descriptions will sound. After you've used Accessibility Inspector to make sure that your user interface elements supply the appropriate information, you should test your application on a device, with VoiceOver turned on, to make sure that it behaves as users expect. See "Using VoiceOver to Test Your Application" (page 30) for some tips on how to do this.

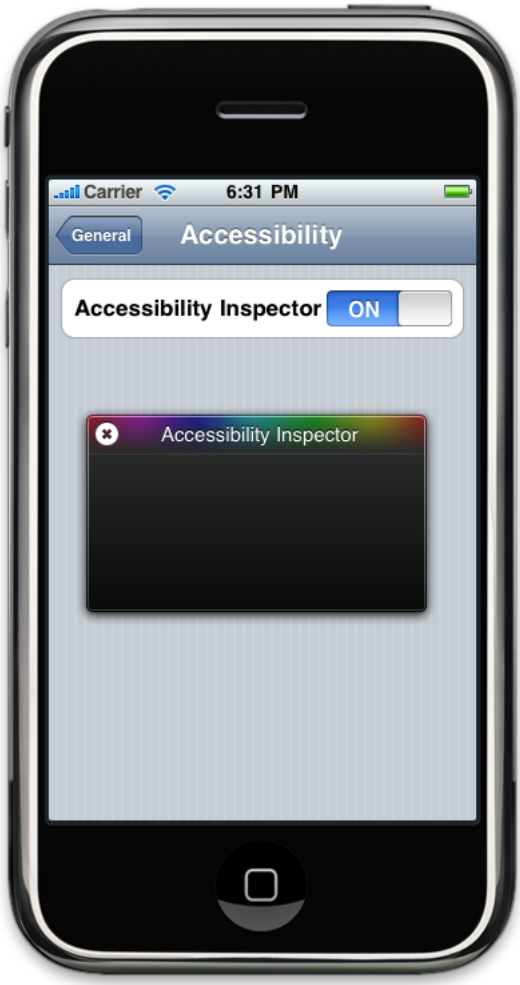
Accessibility Inspector runs in iPhone Simulator and allows you to see at a glance whether an element is accessible. To start Accessibility Inspector, follow these steps:

1. Run your application in iPhone Simulator (for more information on how to do this, see Using iPhone Simulator).
2. In the simulated device environment, click the Home button to reveal the Home screen.
3. Open Settings and go to General > Accessibility.
4. Slide the Accessibility Inspector switch control to on. After you do this, Accessibility Inspector remains active until you turn it off, even if you quit iPhone Simulator and restart it.

After you perform these steps, you should see the Accessibility Inspector panel, as shown in Figure 3-1.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Figure 3-1 Turn on the Accessibility Inspector in Settings



If this is the first time you've used Accessibility Inspector, you might also see an alert that warns you about the different interaction model the inspector uses (this interaction model is described next). After you've read the alert, click OK to dismiss it.

When you run your application in iPhone Simulator, a single click simulates a single tap, and scrolling with the mouse or keyboard simulates flicking or dragging with the finger. But when Accessibility Inspector is active, a single click focuses the inspector on an element; it does not simulate a tap on the element. To simulate a tap on an application element while Accessibility Inspector is active, double-click the element. When Accessibility Inspector focuses on an element, it draws a shaded box around the element, as shown in Figure 3-2.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Figure 3-2 Accessibility Inspector draws a rectangle around the selected element

To simulate scrolling, you must first deactivate Accessibility Inspector. Then, you can scroll as needed and reactivate the inspector when you've reached the desired area in your application. To deactivate or reactivate Accessibility Inspector, click the close control in the upper-left corner of the panel (the close control looks like a circle with an "X" in it). Note that clicking this control does not turn off Accessibility Inspector; to turn it off, you need to set the General > Accessibility setting to off.

When Accessibility Inspector is not active, it appears as it does in Figure 3-3 and it does not affect the way you interact with any simulated application features in iPhone Simulator.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Figure 3-3 The inactive appearance of Accessibility Inspector

Using VoiceOver to Test Your Application

It's a good idea to test your application using VoiceOver, because it allows you to experience the application in the same way that VoiceOver users will experience it. Using VoiceOver to run your application can expose problem areas, for example, confusing labels, unhelpful hints, and unreachable elements, that make your application less accessible.

VoiceOver is a sophisticated application that provides many powerful features to users with disabilities. For example, VoiceOver users can use an invisible dial, known as the rotor control, to change the results of some gestures on the fly. Although you don’t need to become an expert VoiceOver user to test your application with it, you do need to know a handful of basic gestures. This section describes how to activate VoiceOver and use it to run your application. (To learn more about using VoiceOver, see *iPhone User Guide*, available at Apple - Support - Manuals.)

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

First, go to Settings > General > Accessibility > VoiceOver and tap the switch control to turn VoiceOver on. If you provide hints for any accessible elements in your application, check to make sure the Speak Hints switch is on (it is on by default). Before leaving VoiceOver settings, make sure the Speaking Rate slider is adjusted to an appropriate value.

After you've turned VoiceOver on, you'll notice that many familiar gestures have different effects. For example, a single tap causes VoiceOver to speak the selected item and a double tap activates the selected item. There are also a couple of new gestures you should learn to make testing your application with VoiceOver easier. The gestures you're most likely to need for testing are listed below:

- **Drag over the screen**. Select and speak each item as you touch it.
- **Tap**. Speak the selected item.
- **Two-finger tap**. Stop speaking the current item.
- **Flick right or left**. Select the next or previous item.
- **Double tap**. Activate the selected item.
- **Two-finger flick up**. Read all accessible items from the top of the screen.
- **Two-finger flick down**. Read all accessible items from the current position.

There are also a few tasks you might need to perform. For example:

- **Enter text on the keyboard**. Flick left or right to select the desired key, then double-tap to enter the character. Alternatively, you can drag your finger over the keyboard until the desired key is selected. Then, while holding the selected key with one finger, tap the screen with another finger to enter the character.

  Flick up or down to move the insertion point forward or backward in the text.
- **Scroll a list or area of the screen**. Flick up or down with three fingers.
- **Adjust a slider**. Flick up or down (with a single finger) to increase or decrease the setting.
- **Unlock iPhone**. Select the Unlock switch, then double-tap the screen.

When an element is selected, VoiceOver draws a black rectangle around it (similar to the shaded rectangle Accessibility Inspector draws), which is called the VoiceOver cursor. While you're testing your application you can use the VoiceOver cursor as another way to make sure elements are being selected as you expect.

To simulate the experience a visually impaired user might have with your application, you can run it with the VoiceOver Screen Curtain in place. When you activate the Screen Curtain, VoiceOver turns the device display off. Testing with the display turned off obliges you to rely on the information VoiceOver speaks and removes the temptation to use your application as a sighted user would. To turn off the display while you use VoiceOver, triple-tap the screen with three fingers. To turn the display back on, perform the same gesture again.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Accessibility Programming Guide for iOS*.

Date	Notes
2010-07-07	Changed the title from "Accessibility Programming Guide for iPhone OS."
2009-05-29	New document that describes how to make an iPhone application accessible to users with disabilities.

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.

Document Revision History

2010-07-07 | © 2010 Apple Inc. All Rights Reserved.