iOSアクセシビリティプログラミングガイド

Developer

目次

図、リスト

はじめに

iOS 3.0から、視覚障害者の方によるiOSベースのデバイスの使用をサポートするVoiceOverを利用できます。デベロッパは、iOS 3.0で導入されたUI Accessibilityプログラミングインターフェイスを使うと、アプリケーションをVoiceOverユーザにとってアクセシブルにできます。VoiceOverは簡単にいうと、アプリケーションのユーザインターフェイスを説明し、音声とサウンドを使用してユーザによるアプリケーション内のビューとコントロールの移動を支援します。Mac OS XのVoiceOverに慣れている方であれば、その経験を利用して、デバイス上のVoiceOverの使用にもすぐに慣れるでしょう。

iOS 3.0以降で実行するiPhoneアプリケーションは、VoiceOverユーザにとってアクセシブルにする必要があります。iOSおよびiOS SDKでは、次のようにしてこの目標を支援しています。

- 標準のUIKitコントロールおよびビューをデフォルトでアクセシブルにする
- iPhoneアプリケーションをアクセシブルにするための簡素化されたプロセスを定義した、UI Accessibilityプログラミングインターフェイスを提供する
- コードでのアクセシビリティの実装と、アプリケーションのアクセシビリティのテストに役立つツールを提供する

iPhoneアプリケーションを開発中もしくはアップデート中であれば、このドキュメントを読み、アプリケーションをVoiceOverユーザにとってアクセシブルにする方法を学んでください。

この書類の構成

このドキュメントは、次の章で構成されています。

- “iPhoneのアクセシビリティ”（5 ページ）では、デバイス上でのVoiceOverの仕組みを簡単に説明し、アプリケーションをアクセシブルにするための各種のプログラミングインターフェイスとツールを紹介します。
- “iPhoneアプリケーションをアクセシブルにする”（10 ページ）では、アプリケーションをVoiceOverユーザにとってアクセシブルにするための指針を詳しく説明します。
- “iPhoneアプリケーションのアクセシビリティのテスト”（28 ページ）では、Accessibility Inspectorを紹介し、それをVoiceOverと一緒に使用してアプリケーションのアクセシビリティをテストする方法を説明します。

iPhoneのアクセシビリティ

iOSベースのデバイスには最初から、すべての人にデバイスを使いやすくするための機能がいくつか含まれていました（Visual Voicemail、メール（Mail）における大きなフォント、Webページ、写真、マップのズーム機能など）。次のアクセシビリティ機能が加わることで、視覚、聴覚、および身体的な障害を持つ人々がさらに簡単にデバイスを使用できるようになります。

- ズーム機能(Zoom)。デバイス画面全体を拡大します。
- 黒地に白(White on Black)。ディスプレイ上の色を反転します。
- モノラルオーディオ(Mono Audio)。左右のチャンネルの音を組み合わせてモノラル信号にし、両側で再生されるようにします。
- 自動修正テキストを読み上げる(Speak Auto-text)。ユーザの入力時にiPhoneが作成するテキストの修正と候補を読み上げます。
- 音声コントロール(Voice Control)。ユーザが音声コマンドを使用して、電話をかけたりiPodの音楽再生を制御したりできます。

さらに、視覚障害を持つユーザはVoiceOverを頼りにデバイスを使用できます。

iPhoneのアクセシビリティとVoiceOver

VoiceOverはAppleの画期的な画面読み上げ技術で、ユーザは画面を見ることなくデバイスを制御できます。VoiceOverは、アプリケーションのユーザインターフェイスとユーザのタッチの間の仲介役となり、アプリケーションの要素とアクションを音声で説明することによりこれを実現します。VoiceOverがアクティブになっていれば、ユーザは誤って連絡先を削除したり電話をかけてしまう心配をする必要はありません。VoiceOverが、ユーザインターフェイスのどこにユーザがいて、何の操作を実行でき、その操作の結果が何であるかを説明してくれるためです。

アプリケーションは、ユーザが操作するユーザインターフェイス要素がすべてアクセシブルであればアクセシビリティに対応していると言えます。ユーザインターフェイス要素は、自身を正しくアクセシビリティ要素としてレポートしていればアクセシブルであると言えます。

ただし有用性を考えると、アクセシブルなユーザインターフェイス要素は画面上の位置、名前、動作、値、タイプについて正確で有益な情報を提供しなければなりません。その情報をVoiceOverが読み上げてユーザに伝えます。iOS SDKには、アプリケーション内のユーザインターフェイス要素を確実にアクセシブルで有用なものにするためのプログラミングインターフェイスとツールが含まれています（詳細については“iPhoneアクセシビリティAPIとツール” （6 ページ）を参照）。

なぜアプリケーションをアクセシブルにする必要があるのか

以下の理由からiPhoneアプリケーションをVoiceOverユーザにとってアクセシブルにする必要があります。

- ユーザ層を拡大する。懸命に努力してすばらしいアプリケーションを作成したのです。より多くのユーザにアプリケーションを利用してもらうチャンスを逃さないでください。
- 画面を見ることなくユーザがアプリケーションを使用できるようにする。視覚障害者の方でもVoiceOverの支援によりアプリケーションを使用することができます。
- アクセシビリティガイドラインへの対応に役立つ。さまざまな政府機関がアクセシビリティのガイドラインを作成しており、iPhoneアプリケーションをVoiceOverユーザにとってアクセシブルにすることは、これらのガイドラインへの適合に役立ちます。
- 正しいことである。

重要なのは、アクセシビリティ対応は、デベロッパが革新的で魅力的なアプリケーションを創造する能力に影響を及ぼすものではないことを認識することです。UI Accessibilityプログラミングインターフェイスを使って、アプリケーションの外観、つまりインターフェイスとメインロジックを変えないための薄い機能レイヤを1つ追加できます。

iPhoneアクセシビリティAPIとツール

iOS 3.0以降には、UI Accessibilityプログラミングインターフェイスが含まれています。この軽量のAPIによりアプリケーションは、ユーザインターフェイスについての説明と、視覚障害の方によるアプリケーションの利用のためにVoiceOverに必要なすべての情報を提供することができます。

UI Accessibilityプログラミングインターフェイスは UIKitの一部で、デフォルトでは標準UIKitのコントロールとビューの上に実装されます。つまり、標準のコントロールとビューを使用すれば、アプリケーションをアクセシブルにするための作業はほとんど終わっていることになります。カスタマイズの度合いによっては、アプリケーションをアクセシブルにするには単に正確で有用な説明をユーザインターフェイス要素に提供するだけでよい場合もあります。

さらに、iOS SDKはアプリケーションのアクセシビリティ対応に役立つ次のツールも提供しています。

- Interface Builderインスペクタペイン。nibファイルの設計時に説明的なアクセシビリティ情報を付加する簡単な方法を提供します。詳細については、“Interface Builderでのカスタム属性情報の定義”（19 ページ）を参照してください。
- Accessibility Inspector。アプリケーションのユーザインターフェイスに埋め込まれたアクセシビリティ情報を表示し、iOS Simulatorでのアプリケーションの実行時にこの情報を確認できるようにします。アプリケーションのアクセシビリティ情報を調べる方法については、“Accessibility Inspectorを使用したアプリケーションのテスト”（28 ページ）を参照してください。

さらに、VoiceOver自身を使用してアプリケーションのアクセシビリティをテストできます。VoiceOverでアプリケーションをテストする方法については、“VoiceOverを使用したアプリケーションのテスト”（31 ページ）を参照してください。

UI Accessibilityプログラミングインターフェイス

UI Accessibilityプログラミングインターフェイスは、2つの非形式プロトコル、1つのクラス、1つの関数、および少数の定数で構成されます。

- `UIAccessibility`非形式プロトコル。`UIAccessibility`プロトコルを実装したオブジェクトは、アクセシビリティステータス（アクセシブルかどうか）をレポートし、オブジェクトについての説明的な情報を提供します。標準のUIKitコントロールおよびビューはデフォルトで`UIAccessibility`プロトコルを実装しています。
- `UIAccessibilityContainer`非形式プロトコル。このプロトコルは、アプリケーションに含まれるいくつかの、またはすべてのオブジェクトを個別の要素としてアクセシブルにするための`UIView`のサブクラスを許可します。これは、ビューに含まれるオブジェクト自体が`UIView`のサブクラスではなく、そのため自動的にアクセシブルになっていない場合に特に便利です。
- `UIAccessibilityElement`クラス。このクラスは、`UIAccessibilityContainer`プロトコルを通じて返すことができるオブジェクトを定義します。`UIAccessibilityElement`のインスタンスを作成して、`UIView`から派生していないオブジェクトや存在しないオブジェクトなど、自動的にアクセシブルでない項目を表すことができます。
- `UIAccessibilityConstants.h`ヘッダファイル。このヘッダファイルには、アクセシビリティ要素が示すことのできる特性、およびアプリケーションが送信できる通知を表す定数が定義されています。

アクセシビリティ属性

アクセシブルなユーザインターフェイス要素を説明する属性がUI Accessibility APIの中核を構成しています。VoiceOverは、ユーザがコントロールやビューにアクセスしたり、やり取りしたりしたときに属性情報を提供します。

属性はまた、もっとも使用される可能性の高いプログラミングインターフェイスのコンポーネントでもあります。それは、属性が1つのコントロールやビューをほかのものから区別するための情報をカプセル化しているためです。標準のUIKitコントロールおよびビューの場合、デフォルトの属性情報がアプリケーションに対して適当であることを確認するだけで済む可能性もあります。カスタムのコントロールおよびビューの場合は、ほとんどの属性情報を提供する必要があるでしょう。

UI Accessibilityプログラミングインターフェイスは次の属性を定義します。

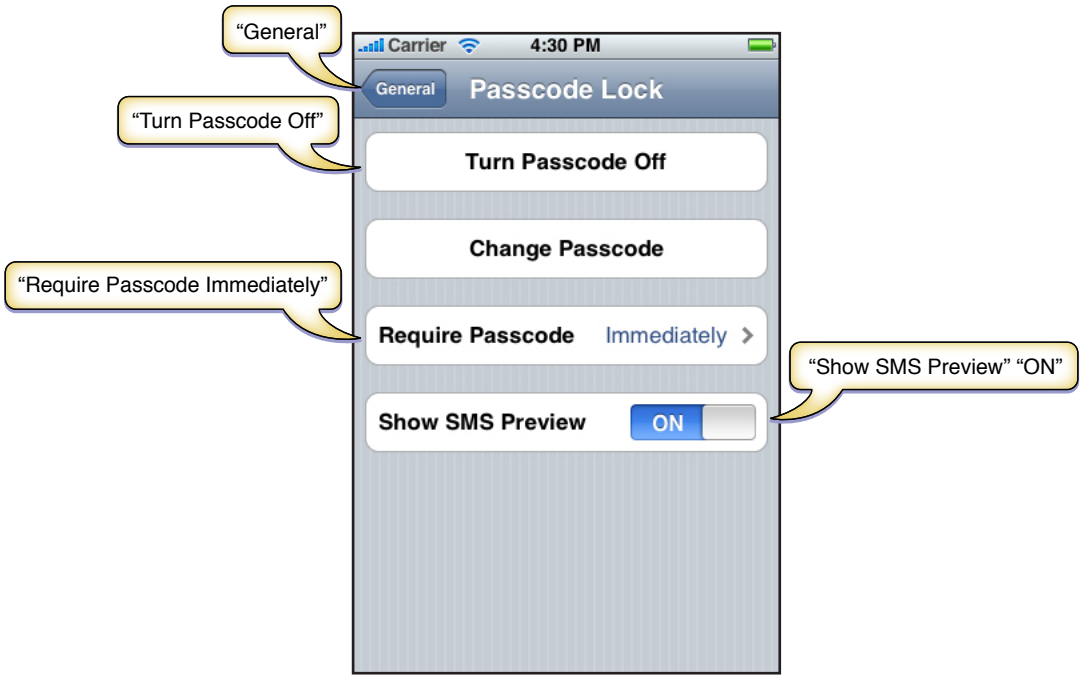
- **Label（ラベル）**。コントロールやビューを簡潔に説明する、短く、ローカライズされた単語またはフレーズですが、要素のタイプは識別しません。例としては、「Add（追加）」や「Play（再生）」などです。
- **Traits（特性）**。それぞれが要素の状態、動作、または使用方法の特徴の1つを説明する個々の特性を、1つ以上組み合わせたものです。たとえば、キーボードのキーのように動作する要素が現在選択されている場合、Keyboard Key特性およびSelected特性の組み合わせで特徴付けることができます。
- **Hint（ヒント）**。要素に対するアクションの結果を説明する、短く、ローカライズされたフレーズです。例としては、「Adds a title（タイトルを追加）」や「Opens the shopping list（ショッピングリストを開く）」などです。
- **Frame（フレーム）**。要素の画面位置とサイズを`CGRect`構造体で表す、要素のフレームの画面座標です。
- **Value（値）**。値がラベルで表されていない場合の、要素の現在の値です。たとえば、スライダのラベルは「Speed（速度）」ですが、その現在の値は「50%」であることなどが考えられます。

それがデフォルトなのか明示的に提供されたのかに関わらず、アクセシビリティ要素は属性の内容を提供します。アクセシビリティ要素は常にFrame属性およびLabel属性の内容を提供します。Frame属性が必須なのは、アクセシビリティ要素は常にユーザインターフェイス内のその位置をレポートできる状態でなければならないためです（`UIView`から派生したオブジェクトはデフォルトでFrame属性を含んでいます）。Label属性が必須なのは、VoiceOverが読み上げるアクセシビリティ要素の名前や説明を含んでいるためです。

アクセシビリティ要素はHint属性およびTraits属性が要素に当てはまらない場合は、その内容を提供する必要はありません。たとえば、アクションを実行しない要素はヒントを提供する必要はありません。

アクセシビリティ要素は、要素の内容が可変で常にラベルで説明することができない場合にのみ、Value属性の情報を提供します。たとえば、電子メールアドレスのテキストフィールドには「Email address（メールアドレス）」というラベルが指定されていることが考えられますが、フィールドの内容はユーザの入力に依存しており、通常は「username@address」という形式になります。図 1-1に、VoiceOverが提供する可能性のあるいくつかの情報を示します。

図 1-1 アクセシブルな要素が提供する情報を読み上げるVoiceOver



VoiceOverユーザは、耳で聞いたラベルやヒントを頼りにしてアプリケーションを使用します。そのため、アプリケーション内のアクセシブルなユーザインターフェイス要素はすべて、正確で参考になる説明を提供することが非常に重要です。標準のUIKitコントロールとビューでは多くの場合、デフォルトの属性情報で適切ですが、これらの要素も調査して確認する必要があります。カスタムコントロールとビューの場合は、属性情報の一部またはすべてをデベロッパが提供しなければならないこともあります。この作業のガイドラインについては、“実用性のあるラベルとヒントの作成”（15 ページ）を参照してください。

iPhoneアプリケーションをアクセシブルにする

iPhoneアプリケーションをアクセシブルにするには、そのユーザインターフェイス要素についての情報をVoiceOverユーザに提供できるようにする必要があります。高いレベルでいうと、次の点を確実にする必要があるということです。

- ユーザがやり取りを行うすべてのユーザインターフェイス要素がアクセシブルである。これには、静的テキストなどの単に情報を提供する要素と同様に、アクションを実行するコントロールも含まれます。
- アクセシブルな要素がすべて、正確で有用な情報を提供している。

これらの基本事項に加えて、Table ViewでのVoiceOverユーザ体験を改善し、アプリケーション内の動的な要素を常に確実にアクセシブルにするためにできることがいくつかあります。

ユーザインターフェイス要素をアクセシブルにする

“iPhoneのアクセシビリティとVoiceOver”（5 ページ）で述べたとおり、ユーザインターフェイス要素は自身をアクセシビリティ要素としてレポートしていればアクセシブルであると言えます。アクセシブルというだけでは、ユーザインターフェイス要素がVoiceOverユーザにとって十分に有用であるとは言えませんが、アプリケーションをアクセシブルにするための第一段階になります。

“iPhoneアクセシビリティAPIとツール”（6 ページ）で述べたように、標準のUIKitコントロールとビューは自動的にアクセシブルになっています。標準のUIKitコントロールのみを使用する場合、アプリケーションをアクセシブルにするためにしなければならない追加作業はほとんどないでしょう。この場合、次のステップはコントロールが提供するデフォルトの属性情報がアプリケーションに対して意味をなしているかどうかを確認することです。方法については、“正確で役に立つ属性情報の提供”（14 ページ）を参照してください。

情報を表示したりユーザがやり取りしたりする必要のあるカスタムビューを作成する場合、これらのビューのアクセシビリティを確認する必要があります。それを終えたら、ビューを使用するユーザをサポートするアクセシビリティ情報を、これらのビューで提供していることを確認する必要があります（“正確で役に立つ属性情報の提供”（14 ページ）を参照）。

アクセシビリティの観点からみると、カスタムビューは単独ビューまたはコンテナビューのどちらかです。**単独ビュー**には、アクセシブルにする必要のあるビューがほかに含まれていません。たとえば、アイコンを表示しボタンのように動作するUIControlのカスタムサブクラスには、ボタン自体のほかにはユーザがやり取りできる要素は含まれていません。単独ビューをアクセシブルにする方法については、“カスタムの単独ビューをアクセシブルにする”（11 ページ）を参照してください。

一方、**コンテナビュー**にはユーザがやり取りできる要素がほかに含まれています。たとえば、幾何学的図形の描画をそれぞれの方法で実行するUIViewのカスタムサブクラスでは、図形がユーザがやり取りできる要素であり、図形はコンテナビューとは異なります。このようなコンテナビュー内の個々の要素は、自動的にはアクセシブルではなく（UIViewのサブクラスではないため）、アクセシビリティ情報を一切提供しません。コンテナビューの内容をアクセシブルにする方法については、“カスタムコンテナビューの内容をアクセシブルにする”（12 ページ）を参照してください。

カスタムの単独ビューをアクセシブルにする

ユーザがやり取りする必要のあるカスタムの単独ビューがアプリケーションに含まれている場合は、そのビューをアクセシブルにする必要があります（単独ビューは、ユーザがやり取りするビューがほかに含まれていないビューであることを思い出してください）。

カスタムの単独ビューをアクセシブルにするには、Interface Builderを使う方法のほかに、プログラミングによってアクセシブルにする方法が2通りあります。1つは、カスタムビューのアクセシビリティステータスをインスタンス化するコードの中で、アクセシビリティステータスを設定する方法です。次のコードにその方法を示します。

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

  /*ここで属性を設定する */
}
```

もう1つはカスタムサブクラスの実装の中で、UIAccessibilityプロトコルのisAccessibilityElementメソッドを実装する方法です。次のコードにその方法を示します。

```
@implementation MyCustomView
  /*ここに属性メソッドを実装する */
```

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

注意 どちらのコード例にもアクセシビリティ属性を設定するコードの必要な箇所にコメントが記述されています。この方法を示したより完全なコードについては、"プログラムによるカスタム属性情報の定義"（21 ページ）を参照してください。

カスタムコンテナビューの内容をアクセシブルにする

アプリケーションで、ユーザがやり取りする要素をほかに含んでいるカスタムビューを表示する場合、含まれている要素を個別にアクセシブルにする必要があります。同時に、コンテナビュー自体はアクセシブルではないことを確認する必要があります。なぜならユーザは、コンテナの中身の要素とやり取りするのであって、コンテナ自身とやり取りするわけではないからです。

これを行うには、カスタムコンテナビューに`UIAccessibilityContainer`プロトコルを実装する必要があります。このプロトコルは、含まれる要素を配列で入手できるようにするメソッドを定義しています。

次のコードは、カスタムコンテナビューの部分的な実装を示したものです。このコンテナビューは、`UIAccessibilityContainer`プロトコルのメソッドが呼び出された場合にのみ、アクセシブルな要素の配列を作成します。したがって、iPhoneアクセシビリティが現在アクティブでない場合、配列は作成されません。

リスト 2-1 カスタムコンテナビューの内容を個別のアクセシビリティ要素としてアクセシブルにする

```
@implementation MultiFacetedView
- (NSArray *)accessibleElements
{
   if ( _accessibleElements != nil )
   {
      return _accessibleElements;
   }
   _accessibleElements = [[NSMutableArray alloc] init];

   /*最初に含まれる要素を表すアクセシビリティ要素を作成し、MultiFacetedViewのコンポーネントとして初期化する */
```

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

    /*ここで最初に含まれる要素の属性を設定する */
    [_accessibleElements addObject:element1];

    /*同様のステップを2番目に含まれる要素に実行する */
    UIAccessibilityElement *element2 = [[[UIAccessibilityElement alloc]
initWithAccessibilityContainer:self] autorelease];

    /*ここで2番目に含まれる要素の属性を設定する */
    [_accessibleElements addObject:element2];

    return _accessibleElements;
}

/*コンテナ自身はアクセシブルではないため、MultiFacetedViewはisAccessiblityElementに対して
NOを返す */
- (BOOL)isAccessibilityElement
{
    return NO;
}

/*以下のメソッドはUIAccessibilityContainerプロトコルメソッドの実装 */
- (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
```

正確で役に立つ属性情報の提供

アクセシブルな要素に対して属性情報を提供するプロセスは次の2つの部分からなります。

- 簡潔、正確、役に立つ情報を練り上げる
- アプリケーション内のアクセシブルな要素が内容を確実に正しくレポートできるようにする

カスタムビューを使用する場合、それらに適切な属性情報をすべて提供する必要があります。参考として、“実用性のあるラベルとヒントの作成”（15 ページ）、“ヒント作成のためのガイドライン”（16 ページ）、および“適切な特性の指定”（18 ページ）を参照してください。

標準のUIKitコントロールとビューのみを使用する場合でも、それらのデフォルトの属性情報のいくつかは、自分のアプリケーションの中でより理にかなったものに改善できると思うことがあります。詳細については、“デフォルトの属性情報の改善”（14 ページ）を参照してください。

標準またはカスタムのUI要素に対してアクセシビリティ属性を提供または変更する必要がある場合、Interface Builderを使用するか（“Interface Builderでのカスタム属性情報の定義”（19 ページ）を参照）、プログラムを使用して（“プログラムによるカスタム属性情報の定義”（21 ページ）を参照）行うことができます。

デフォルトの属性情報の改善

標準のUIKitコントロールおよびビューの組み込みアクセシビリティの一部として、iOSはVoiceOverユーザにこれらの要素を説明するデフォルトの属性情報も提供しています。ほとんどの場合、標準のコントロールとビューを使用するアプリケーションにとってこの情報は適切です。ただし、カスタムの属性情報を提供するほうがVoiceOverユーザのアプリケーションの体験を改善できる場合があります。

- システムで提供されるアイコンやタイトルを表示する標準のUIKitコントロールやビューを使用する場合は、それが意図した目的に沿って使用されていることを最初に確認します（詳細については、『*iPhoneHumanInterfaceGuidelines*』を参照）。次に、デフォルトのLabel属性が、アプリケーションのコントロールまたはビューを使用した場合の結果を正確に伝えているかを判断します。そうでない場合、Hint属性を提供することを検討します。

  たとえば、`UIBarButtonItem`オブジェクト内のシステム指定の“Add (+)”アイコンを使用してナビゲーションバーに「Add（追加）」ボタンを配置する場合、デフォルトのLabel属性である“Add”が自動的に設定されます。このボタンをアクティブ化したときにユーザがどのアイテムを追加して

いるのかが常に明白であれば、Hint属性を提供する必要はありません。しかし混乱の可能性がある場合、アプリケーションでそのコントロールを使用した場合の結果を説明する、「Adds an account（アカウントを追加）」または「Adds a comment（コメントを追加）」のように、独自のヒントを提供することを検討する必要があります。

- `UIButton`オブジェクトのような標準のUIKitビューにカスタムのアイコンや画像を表示する場合、それを説明するカスタムのLabel属性を提供する必要があります。

実用性のあるラベルとヒントの作成

VoiceOverユーザがアプリケーションを実行するとき、ユーザはVoiceOverが読み上げる説明を頼りに、アプリケーションが何を実行するのかと、その使用方法を理解します。これらの説明がVoiceOverユーザにとってのアプリケーション体験の大部分を占めるため、説明は可能な限り正確で有用であることが重要です。このセクションのガイドラインは、障害者の方がアプリケーションを容易に操作し楽しめるようなラベルとヒントの作成に役立ちます。

ラベル作成のためのガイドライン

Label属性はユーザインターフェイス要素を識別します。アクセシブルなユーザインターフェイス要素（標準およびカスタム）はすべて、Label属性の内容を提供しなければなりません。

> **注意** テーブルの行もLabel属性を持つことができます。ただし、テーブルの行のラベル作成のガイドラインはその他のタイプのコントロールやビューのラベル作成のガイドラインとは異なります。テーブルの行に対して実用性のあるラベルを作成する方法については、“Table Viewのアクセシビリティの改善”（23 ページ）を参照してください。

ラベルが何を伝えるべきかを判断する良い方法は、目の見えるユーザがそれを見ただけでアプリケーションについて何を推測するかを考えることです。優れたユーザインターフェイスを設計していれば、目の見えるユーザはそのタイトルを読んだりアイコンを理解したりして、コントロールまたはビューが現在のアプリケーションで何を実行するかを知ることができるはずです。これが、Label属性でVoiceOverユーザに提供しなければならない情報です。

カスタムのコントロールやビューを提供したり、標準のコントロールやビューにカスタムアイコンを表示したりする場合は、次のようなラベルを提供する必要があります。

- **要素を非常に簡潔に説明している。**「Add（追加）」、「Play（再生）」、「Delete（削除）」、「Search（検索）」、「Favorite（お気に入り）」、「Volume（音量）」など、1つの単語で構成されるのが理想的です。

1つの単語で要素を識別でき、その使用方法が現在のコンテキストにおいて明白になるようにアプリケーションを設計することを目指してください。しかし、場合によっては短いフレーズを使用して適切に要素を識別する必要もあります。このような場合は、「Play Music（音楽を再生）」、「Add Name（名前を追加）」、「Add to Event（イベントに追加）」のように非常に短いフレーズを作成します。

- **コントロールやビューのタイプを含まない。**タイプの情報は要素のTraits属性に含まれており、ラベルで重複するべきではありません。

  たとえば、「Add（追加）」ボタンのラベルにコントロールのタイプを含めると、VoiceOverユーザはそのコントロールにアクセスするたびに「Add button button（追加ボタンボタン）」と聞くことになります。この体験はすぐに煩わしくなり、ユーザはアプリケーションを使いたくなくなるでしょう。
- **先頭の単語が大文字で始まる（英語の場合）。**これによりVoiceOverは適切な抑揚でラベルを読み上げます。
- **ピリオドを付けない。**ラベルは文章ではないため最後にピリオドを付けるべきではありません。
- **ローカライズされている。**アクセシビリティ属性文字列を含むすべての文字列をローカライズすることで、できるだけ多くのユーザがアプリケーションを利用できるようにしてください。通常、VoiceOverはユーザが「言語環境(International)」設定で指定した言語を使用して読み上げます。

ヒント作成のためのガイドライン

Hint属性は、コントロールまたはビューに対してアクションを実行した場合の結果を説明します。アクションの結果が要素のラベルからでは分かりにくい場合にのみヒントを提供します。

たとえば、アプリケーション内に「Play（再生）」ボタンを提供する場合、ボタンが表示される状況によって、それをタップすると何が起きるかユーザが簡単に理解できるようにする必要があります。しかし、リスト内の曲名をタップすることで曲を再生できるようにする場合は、この結果を説明するヒントを提供する必要があるかも知れません。リストアイテムのラベルはアイテム自身（この場合は曲名）の説明であって、ユーザがタップした場合に何が起きるかを説明しているわけではないからです。

> **注意**　VoiceOverユーザは、デバイスの「VoiceOver」設定のオプションを選択して利用可能なヒントを聞くかどうかを選択できます。デフォルトでは、ヒントの読み上げは有効になっています。

コントロールやビューに対するユーザのアクションの結果がラベルでは明白に伝わらない場合、次のようなヒントを作成します。

- **結果を非常に簡潔に説明する。**ヒントが必要なコントロールやビューは少数ですが、提供する必要のあるヒントはできるだけ簡潔にすることを目指します。そうすることで、要素を使用する前にユーザが聞き取りに費やさなければならない時間を減らすことができます。

  とはいえ、簡潔にするために明確さや正しい文法を犠牲にするのは避けてください。たとえば、「Adds a city（都市を1件追加します）」を「Adds city（都市を追加します）」に変更してもヒントの長さが大幅に減るわけではありませんが、ぎこちなく聞こえるうえ、少し曖昧になります。

- **動詞で始め、主語を省略する**動詞の3人称単数形（「Plays」など）を用い、不定形（「Play」など）は使わないでください。ヒントが命令のように聞こえてしまうため、不定形の動詞は避けるようにします。たとえば、ユーザに対しては「Play the song（曲を再生しろ）」ではなく、要素をタップすると「Plays the song（曲を再生します）」と伝える必要があります。

  適切な言い回しを考えるときには、コントロールの使用方法を友人に説明していると想像します。「Tapping this control plays the song（このコントロールをタップすると曲を再生します）」のような説明をするでしょう。多くの場合、このような文章では2つ目のフレーズ（この場合では、「Plays the song（曲を再生します）」）をヒントとして使用できます。

- **先頭が大文字の単語で始まり、ピリオドで終了する。（英語の場合）**ヒントは文章ではなくフレーズですが、ヒントをピリオドで終了することによりVoiceOverが適切な抑揚でそれを読み上げるようになります。

- **アクションやジェスチャの名前を含まない。**ヒントではユーザにアクションの実行方法は伝えず、アクションを実行すると何が起こるかを伝えます。したがって、「Tap to play the song（曲を再生するにはタップします）」、「Tapping purchases the item（タップするとアイテムを購入します）」、または「Swipe to delete the item（スワイプしてアイテムを削除します）」といったヒントを作成してはいけません。

  これが特に重要なのは、VoiceOverユーザはVoiceOver固有のジェスチャを使用してアプリケーション内の要素を操作できるためです。ヒントに異なるジェスチャが含まれていると、とても紛らわしくなります。

- **コントロールやビューの名前を含まない。**ユーザはこの情報をLabel属性から取得するため、ヒントの中で繰り返すべきではありません。したがって、「Save saves your edits（「保存」は編集内容を保存します）」または「Back returns to the previous screen（「戻る」は前の画面に戻ります）」のようなヒントを作成してはいけません。

- **コントロールやビューのタイプを含まない。**たとえば、コントロールやビューがボタンのように動作するのか、検索フィールドのように動作するのかユーザはすでにわかっています。なぜなら、その情報は要素のTraits属性から利用できるためです。このため、「Button that adds a name（名前を追加するボタン）」や「Slider that controls the scale（縮尺を調整するスライダ）」といったヒントを作成してはいけません。

- **ローカライズされている。**アクセシビリティラベルと同様、ヒントもユーザの好みの言語で利用可能にするべきです。

適切な特性の指定

Traits属性には1つ以上の個別の特性が含まれており、それらを組み合わせて、アクセシブルなユーザインターフェイス要素の動作を説明します。いくつかの個別の特性を組み合わせて1つの要素を説明することができるため、要素の動作を正確に表すことができます。

> **注意** 個々の特性はOR演算子を使用して結合します。このドキュメントのコード例以外の部分では、結合方法は明記せずに、短く「結合する」や「結合」という表現を使用しています。

ボタンやテキストフィールドなどの標準のUIKitコントロールは、Traits属性用のデフォルトの内容を用意しています。アプリケーションで標準のUIKitコントロールのみを使用している（かつその動作をまったくカスタマイズしていない）場合、これらのコントロールのTraits属性を変更する必要はありません。

標準コントロールの動作をカスタマイズした場合、コントロールのデフォルトの特性を新しい特性と結合する必要があるかも知れません。カスタムのコントロールまたはビューを作成した場合、要素のTraits属性の内容を指定する必要があります。

UI Accessibilityプログラミングインターフェイスは12の個別の特性を定義しており、それらのいくつかは結合することができます。特性の中には、特定のタイプのコントロール（ボタンなど）または特定のタイプのオブジェクト（画像など）の動作により要素の動作を特定することで要素を特徴付けるものがあります。あるいは、要素が示す特定の動作（音の再生など）を説明することによって要素を特徴付ける特性もあります。

アプリケーション内の要素を特徴付けるには次の特性を使用します。

- ボタン
- Link
- Search Field
- Keyboard Key
- Static Text
- Image
- Plays Sound
- Selected
- Summary Element
- Updates Frequently
- Not Enabled

- なし

一般に、コントロールに対応する特性は動作を説明する特性とうまく結合させることができます。たとえば、Button TraitをPlays Sound Traitと結合し、ボタンのように動作してタップされると音を再生するカスタムコントロールを特徴付けることができます。

ほとんどの場合、特定のコントロールに対応する特性は、相互に排他的になるように考慮するべきです。具体的には、Button、Link、Search Field、およびKeyboard Keyの特性です。つまり、これらの特性を複数使用してアプリケーション内の単一の要素を特徴付けるべきではありません。代わりに、これら4つの特性のうち、どれがアプリケーション内の要素にもっとも近く対応するかを考えます。次に、要素にほかの動作がある場合は、動作を表す特性のうちの1つを、最初の特性と結合することができます。

たとえば、ユーザのタップに応答して、iPhone上のSafariでリンクを開く画像をアプリケーション内に表示するとします。この要素をImage TraitおよびLink Traitを結合して特徴付けることができます。もう1つの例としては、タップされるとほかのキーボードキーを変更するキーボードのキーがあります。この要素をKeyboard Key TraitおよびSelected Traitを結合して特徴付けることができます。

特性によってコントロールがどのように特徴付けられているかの例をいくつか見るには、Accessibility Inspectorを使うと、標準コントロールに設定されているデフォルトの特性を確認できます。Accessibility Inspectorの使い方の詳細については、“Accessibility Inspectorを使用したアプリケーションのテスト”（28 ページ）を参照してください。

Interface Builderでのカスタム属性情報の定義

iOS SDK 3.0をインストールすると、アプリケーションをアクセシブルにするのに役立つ機能が含まれたバージョンのInterface Builderがインストールされます。アプリケーションに標準のUIKitコントロールおよびビューが含まれている場合、すべてのアクセシビリティ対応作業をInterface Builderで行うことができるかも知れません。

Interface Builderを使用して、要素のアクセシビリティステータスを設定し、Label、Hint、およびTraits属性にカスタムコンテンツを指定できます。これを行うには、nibファイルでユーザインターフェイス要素を選択し、「Identity」インスペクタを開きます。インスペクタの「Accessibility」セクションを開くと、図 2-1に示すようなものが表示されます。

図 2-1 Interface Builderに表示される、標準のText Fieldのデフォルトアクセシビリティ情報

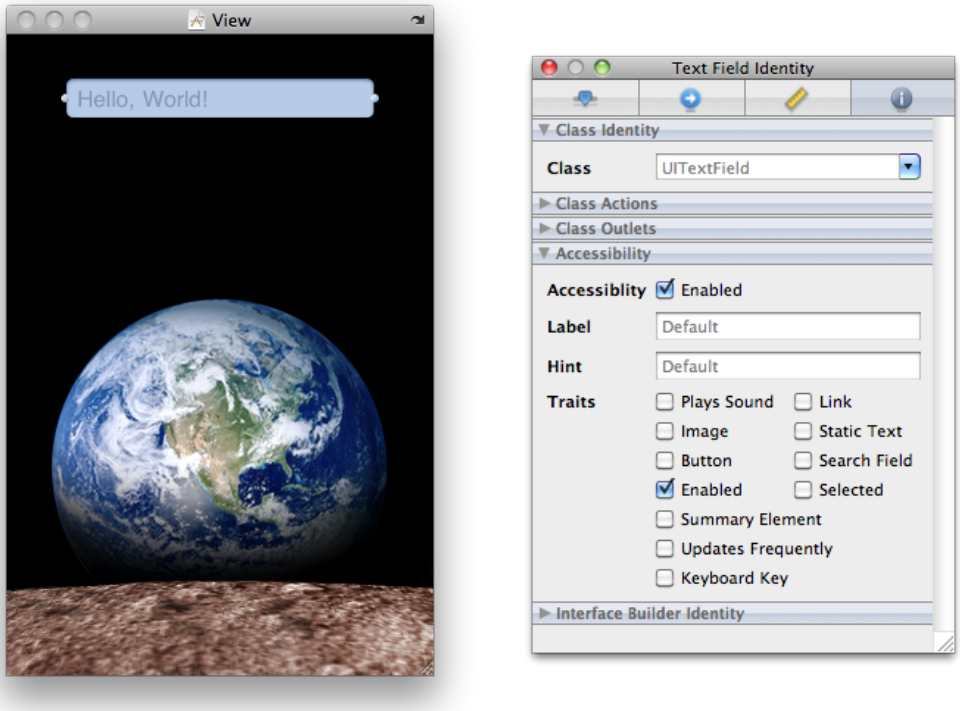


図 2-1に示すように、nibファイルで使用される標準のText Fieldはデフォルトでアクセシブルになっており、Label、HintおよびTraits属性のデフォルトの情報を含んでいます（プレースホルダテキストを表示するText Fieldの場合、デフォルトのLabelはプレースホルダテキストです）。図 2-2に示すように、これらのデフォルトの値はどれも「Identity」インスペクタで新しい情報を設定して変更できます（図

2-2は、Accessibility InspectorがText Fieldのアクセシビリティ情報をどのように表示するかも示しています。Accessibility Inspectorの詳細については、“Accessibility Inspectorを使用したアプリケーションのテスト”（28 ページ）を参照してください）。

図 2-2 Interface Builderでのアクセシビリティ情報の指定

プログラムによるカスタム属性情報の定義

属性のカスタム情報はプログラミングによって指定することもできます。Interface Builderをまったく使用しない場合や、Interface Builderを使用する代わりにコードでビューを生成している場合、この方が便利かも知れません。

“カスタムの単独ビューをアクセシブルにする”（11 ページ）に説明されているとおり、アクセシビリティ情報はビューのサブクラスの実装の中、またはビューをインスタンス化するコード内で設定できます。どちらのテクニックも有効ですが、インスタンス化のコードで属性を設定するのではなく、サブクラスで属性メソッドを実装する方が良い理由が1つあります。“時刻”のような動的な、または頻

繁に変化するデータをビューで表示する場合、サブクラスメソッドを実装して必要なときに最新のデータを返すようにする必要があります。そのような場合、サブクラスをインスタンス化したときに属性を設定するだけでは返されるデータは古くなっている可能性が高くなります。

このセクションのコード例は、“カスタムの単独ビューをアクセシブルにする”（11 ページ）のコードに属性特有のメソッドを追加したものです。たとえば、サブクラス内でアクセシビリティメソッドを実装する場合、リスト 2-2のようなコードを記述します。

リスト 2-2　カスタムサブクラス実装内での属性情報の設定

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

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

/*ボタンのように動作するカスタムビュー */
- (UIAccessibilityTraits)accessibilityTraits
{
   return UIAccessibilityTraitButton;
}

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

カスタムビューをインスタンス化するコード内で`UIAccessibility`プロトコルのプロパティ設定メソッドを使用する場合は、リスト 2-3のようなコードを記述します。

リスト 2-3　カスタムサブクラスオブジェクトをインスタンス化するコード内での属性情報の指定

```
@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)];
}
```

Table Viewのアクセシビリティの改善

アプリケーションで表示するTable Viewの各セルに、テキスト以外の（またはテキストに加えて）アイテムを含める場合、それをよりアクセシブルにするためにできることがいくつかあります。同様に、Table Viewの行ごとに複数の情報を表示する場合、情報を1つの理解しやすいラベルに集約することでVoiceOverユーザの体験を改善することができます。

> **注意**　テーブルのセルに、ディスクロージャインジケータ、詳細ディスクロージャボタン、または削除コントロールなどの標準のTable View要素のいずれかが含まれている場合、これらの要素をアクセシブルにするために必要な作業はありません。ただし、テーブルのセルにスイッチやスライダなど、ほかの種類のコントロールが含まれている場合は、これらの要素が適切にアクセシブルになっていることを確認する必要があります。

アプリケーションのテーブルのセルに異なる要素の組み合わせが含まれている場合、ユーザが各セルを1つの部品として操作するのか、またはセル内の個々の要素を操作するのかを決めます。セル内の個々の要素にアクセスする必要がある場合は、次のことが必要です。

- 個々の要素をそれぞれアクセシブルにする。
- テーブルのセル自体はアクセシブルにしない。
- セルの内容全体を簡潔に説明し、その説明をセルのLabel属性として使用する。この場合、ラベルはセル内のアクセシブルな要素の1つとして見なされることに注意します。

テキストやコントロールなどの複数のアイテムを含むテーブルのセルは、UI Accessibilityプログラミングインターフェイスで定義されているコンテナビューの基準に当てはまることに気が付いたかも知れません。しかし、テーブルのセルは自動的にコンテナとして指定されているため、セルをコンテナビューとして指定したり、`UIAccessibilityContainer`プロトコルのメソッドを実装したりする必要はありません。

テーブルに、それぞれ大量の情報を提供するセルが含まれている場合は、これらの中からの情報を組み合わせてLabel属性にすることを検討するべきです。こうすれば、VoiceOverユーザはセルの内容の意味を1つのジェスチャで得ることができ、個々の情報にそれぞれアクセスする必要がなくなります。

この仕組みを示すのに良い例が、組み込みの「株価(Stocks)」アプリケーションです。会社名、現在の株価、および価格の変動を別々の文字列として提供する代わりに、「株価(Stocks)」ではこの情報をラベル内で結合し、次のように読み上げられます。"Apple Inc., $125.25, up 1.3%."ラベルのカンマに注目してください。別々の情報をこのように結合する場合、カンマを使用してVoiceOverにフレーズ間で短く一時停止させることができ、ユーザにとって情報をより理解しやすいものにすることができます。

以下のコードでは2つの別々の要素のラベル情報を結合し、両方を説明する1つのラベルにする方法を示しています。

```
@implementation CurrentWeather

/*気象情報を提供するビュー。それぞれのラベルを提供する都市(City)サブビューと気温(Temperature)
サブビューが含まれている */

- (NSString *)accessibilityLabel

{

   NSString *weatherCityString = [_weatherCity accessibilityLabel];

   NSString *weatherTempString = [_weatherTemp accessibilityLabel];


   /*都市(City)と気温(Temperature)情報を結合し、VoiceOverユーザが気象情報を1つのジェスチャ
で取得できるようにする */


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

}

@end
```

動的要素をアクセシブルにする

アプリケーション内のユーザインターフェイス要素が動的に変化する場合、提供するアクセシビリティ情報が正確かつ最新であることを確認する必要があります。また、アプリケーション画面のレイアウトに変更が発生した際には通知を送信し、VoiceOverが新しいレイアウトでのユーザの移動をサポートできるようにする必要もあります。UI Accessibilityプログラミングインターフェイスは、このような変更が画面に発生した際に使用できる2種類の通知を提供しています（これらの通知の詳細については、『*UIAccessibility Protocol Reference*』の「Notifications」を参照してください）。

ユーザインターフェイス要素が、アプリケーションのその他の条件に応じて異なる状態になる場合には、要素がとるそれぞれの状態について正しいアクセシビリティ情報を返すロジックをコードに追加する必要があります。このような変化はユーザアクションの結果として発生することがあるため、サブクラスをインスタンス化するコードではなく、サブクラスの実装にロジックを追加するのが最良です。

次のコードは動的な状態の変化をどのように処理し、画面のレイアウトに変更があった場合にどのように通知を送信するかを示しています。コード例はカスタムのKeyboard Keyのように動作する、`UIView`サブクラスの実装を表しています。Keyのアクセシビリティラベルは、インスタンスが表しているのが英数字なのか英数字以外の文字なのか、そしてシフトキーが現在選択されているかどうかによって変化します。Keyはまた、どんな種類のKeyboard Keyを表しているのか、そしてそれが現在選択されているかどうかに応じて異なるアクセシビリティ特性を返します。リスト 2-4のコードは、キーボードの状態を問い合わせる複数のメソッドが存在することを前提としています。

リスト 2-4　現在の状態の正しいアクセシビリティ情報の返却およびレイアウト変更通知の送信

```
@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;

    /*これがShiftキーで選択済みの場合、ユーザはShiftが現在有効であることを知る必要がある */
    if ( [self isShiftKey] && [self isSelected] )
    {
        traits |= UIAccessibilityTraitSelected;
    }

    return traits;
}

- (void)keyboardChangedToNumbers
{
    /*数値キーボードの変更を実行するコードをここに記述 */

    /*画面レイアウトの変更の通知を送信 */

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

テキスト形式でないデータをアクセシブルにする

場合によっては、アクセシビリティの仕組みに自動的には準拠しないデータをアプリケーションで表示することがあります。たとえば画像を表示する場合、その説明をアクセシビリティラベルに設定し、画像が伝える情報をVoiceOverユーザが理解できるようにしなければなりません。または、星を表示する評価システムのように情報をグラフィカルに提供する場合、グラフィカルな表現の裏にある意味をアクセシビリティラベルが伝えていることを確認しなければなりません。

次のコードでは、アイテムの評価に対応した数の星を描くカスタムビューの例を使用しています。コードは描く星の数に応じて、ビューがどのように適切なアクセシビリティラベルを返しているかを示しています。

```
@implementation RatingView
/*ここにほかのサブクラス実装のコード */


- (NSString *)accessibilityLabel
{
   /* _starCountは描く星の数を保持するインスタンス変数 */
   NSInteger starCount = _starCount;
   if ( starCount == 1 )
   {
      ratingString = NSLocalizedString(@"rating.singular.label", nil); // ここでは、
ratingStringは「star」
   }
   else
   {
       ratingString = NSLocalizedString(@"rating.plural.label", nil); // ここでは、
ratingStringは「stars」
   }


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

iPhoneアプリケーションのアクセシビリティのテスト

iOS OS 3.0以降には、アプリケーションのアクセシビリティをテストする2つの補完的な方法があります。

- iOS SimulatorのAccessibility Inspector
- デバイス上のVoiceOver

最良の結果を得るためには、両方の方法でアプリケーションをテストする必要があります。

Accessibility Inspectorを使用したアプリケーションのテスト

Accessibility Inspectorは、アプリケーション内のアクセシブルなそれぞれの要素についてアクセシビリティ情報を表示します。Accessibility Inspectorを使用して、VoiceOverとアプリケーション内のアクセシブルな要素とのやり取りをシミュレートし、提供される情報を確認できます。

> **注意** Accessibility Inspectorは開発中のアプリケーションのアクセシビリティをテストするには理想的ですが、VoiceOverを使ったアプリケーションのテストの代わりにはなりません。1つの理由としては、Accessibility Inspectorはアクセシビリティ情報を読み上げないので、要素の説明がどのように聞こえるのかを聞くことができません。Accessibility Inspectorを使用してユーザインターフェイス要素が適切な情報を提供していることを確認したら、デバイス上でVoiceOverを有効にしてアプリケーションをテストし、ユーザの期待通りに動作することを確認します。これを行うためのいくつかのヒントは、"VoiceOverを使用したアプリケーションのテスト"（31 ページ）を参照してください。

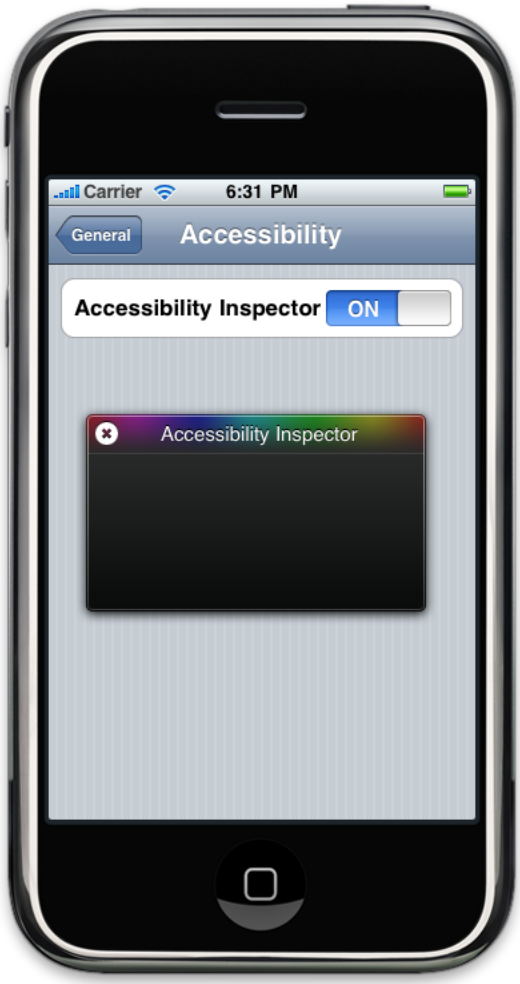
Accessibility InspectorはiOS Simulator上で実行でき、要素がアクセシブルかどうかを一目で確認できます。Accessibility Inspectorを開始するには、次の手順に従ってください。

1. iOS Simulatorでアプリケーションを実行します（詳細については、『"Using iOS Simulator"』を参照してください）。
2. シミュレートされたデバイスの環境で、「ホーム(Home)」ボタンをクリックしてホーム(Home)画面を表示します。
3. 「設定(Settings)」を開き、「一般(General)」＞「アクセシビリティ(Accessibility)」と進みます。

4. 「Accessibility Inspector」スイッチコントロールを「オン(ON)」へスライドします。この後、iOS Simulatorを終了して再起動しても、Accessibility Inspectorはオフに切り替えられるまでアクティブな状態が続きます。

以上の手順を実行すると、図 3-1に示すような、Accessibility Inspectorパネルが表示されます。

図 3-1 「設定(Settings)」でのAccessibility Inspectorの有効化



Accessibility Inspectorを初めて使用する場合、使用される異なる対話モデルについての警告が表示される場合もあります（この対話モデルは次に説明します）。警告が表示されたら、「OK」をクリックして閉じます。

iOS Simulatorでアプリケーションを実行する場合、シングルクリックはタップ1回をシミュレートし、マウスやキーボードによるスクロールはフリックまたは指でのドラッグをシミュレートします。ただしAccessibility Inspectorがアクティブな場合は、シングルクリックによって要素にフォーカスが置かれます。要素に対するタップはシミュレートしません。Accessibility Inspectorがアクティブな場合にアプ

リケーション要素へのタップをシミュレートするには、要素をダブルクリックします。Accessibility Inspectorが要素にフォーカスすると、図 3-2に示すように、要素の周りに影付きボックスを描画します。

図 3-2 選択した要素の周りに矩形を描画するAccessibility Inspector

スクロールをシミュレートするには、最初にAccessibility Inspectorを非アクティブにする必要があります。次に、必要なだけスクロールし、アプリケーションで希望する位置に到達したらAccessibility Inspectorを再びアクティブにします。Accessibility Inspectorを非アクティブ化または再アクティブ化するには、パネルの上部左の隅のCloseコントロールをクリックします（Closeコントロールは中に"X"のある円のように見えます）。このコントロールをクリックしてもAccessibility Inspectorは終了しません。終了するには「一般(General)」＞「アクセシビリティ(Accessibility)」の設定を「オフ(OFF)」にします。

Accessibility Inspectorがアクティブでない場合は、図3-3にように表示され、iOS Simulatorでシミュレートされているどのアプリケーション機能に対する操作にも影響しません。

図 3-3 Accessibility Inspectorの非アクティブな外観

VoiceOverを使用したアプリケーションのテスト

VoiceOverを使用してアプリケーションをテストするのは、VoiceOverユーザが体験するのと同じ方法でアプリケーションを体験できるため良い考えです。VoiceOverを使用してアプリケーションを実行することにより、アプリケーションのアクセシビリティを下げるような問題のある領域（たとえば分かりにくいラベル、参考にならないヒント、到達できない要素など）を明らかにできます。

VoiceOverは、障害を持つユーザにとって、パワフルで多機能な洗練されたアプリケーションです。たとえば、VoiceOverユーザはローターコントロールとして知られる非表示のダイヤルを使用して、いくつかのジェスチャの結果をその場で変更できます。アプリケーションをテストするためにVoiceOver

ユーザのエキスパートになる必要はありませんが、いくつかの基本的なジェスチャは知っておく必要があります。このセクションではVoiceOverをアクティブ化する方法と、これを使用してアプリケーションを実行する方法を説明します（VoiceOverの使用方法の詳細については、アップルーサポートーマニュアルで入手できる、『*iPhoneユーザガイド*』を参照してください）。

最初に、「設定(Settings)」＞「一般(General)」＞「アクセシビリティ(Accessibility)」＞「VoiceOver」と進み、スイッチコントロールをタップしてVoiceOverを起動します。アプリケーションのアクセシブルな要素に1つでもヒントを提供している場合、読み上げのヒントスイッチがオン(ON)であることを確認します（デフォルトは「オン(ON)」です）。「VoiceOver」設定を終了する前に、話す速さ(Speaking Rate)のスライダが適当な値に調節されていることを確認します。

VoiceOverの起動後、多くの使い慣れたジェスチャが異なる効果を持っていることに気が付きます。たとえば、シングルタップによりVoiceOverは選択した項目を読み上げ、ダブルタップは選択したアイテムをアクティブ化します。また、VoiceOverを使用したアプリケーションのテストをより簡単にするために学ぶべき新しいジェスチャがいくつかあります。テストにもっとも必要と思われるジェスチャは以下のとおりです。

- **画面上のドラッグ。**タッチした各項目を選択し読み上げます。
- **タップ。**選択した項目を読み上げます。
- **2本の指でタップ。**現在の項目の読み上げを停止します。
- **右または左にフリック。**次または前の項目を選択します。
- **ダブルタップ。**選択した項目を有効にします。
- **2本の指で上にフリック。**画面の一番上からすべてのアクセシブルな項目を読み上げます。
- **2本の指で下にフリック。**現在の位置からすべてのアクセシブルな項目を読み上げます。

実行しなければならない可能性のある作業もいくつかあります。たとえば次のように宣言します。

- **キーボードでテキストを入力する。**左右どちらかにフリックして使用するキーを選択し、ダブルタップして文字を入力します。または、使用するキーが選択されるまでキーボードの周囲を指でドラッグすることもできます。そして、選択したキーを1本の指で押したまま、別の指で画面をタップして文字を入力します。

  挿入ポイントをテキストの前後に移動させるには上または下にフリックします。
- **画面のリストまたは領域をスクロールする。**3本の指で上または下にフリックします。
- **スライダの調節。**設定を増減させるには、（1本の指で）上または下にフリックします。
- **iPhoneのロック解除。**ロック解除(Unlock)スイッチを選択し、画面をダブルタップします。

要素が選択されると、VoiceOverはVoiceOverカーソルと呼ばれる黒い長方形でその周りを囲みます（Accessibility Inspectorが描く影付きボックスに似ています）。アプリケーションをテストする際に、要素が意図したとおりに選択されることを確認するための1つの手段としてVoiceOverカーソルを使用することができます。

視覚障害を持つユーザによるアプリケーションの利用を擬似体験するために、VoiceOverスクリーンカーテン(Screen Curtain)を使用して実行できます。スクリーンカーテン(Screen Curtain)を有効にすると、VoiceOverはデバイスの画面をオフにします。画面をオフにしてテストすることによりVoiceOverが読み上げる情報を頼りにしなければならなくなり、目の見えるユーザがするようにアプリケーションを使用する誘惑が排除されます。VoiceOverの使用中に画面をオフにするには、画面を3本の指でトリプルタップします。画面をオンに戻すには、同じジェスチャを再度実行します。

書類の改訂履歴

この表は「*iOSアクセシビリティプログラミングガイド*」の改訂履歴です。

日付	メモ
2012-02-16	欠落していたreturn文をコードに追加しました。
2011-07-20	細かな誤字を訂正しました。
2010-07-07	『iPhone OSアクセシビリティプログラミングガイド』からドキュメント名を変更しました。
2009-05-29	障害を持つユーザにとってiPhoneアプリケーションをアクセシブルにする方法を記述した新規ドキュメント。

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/