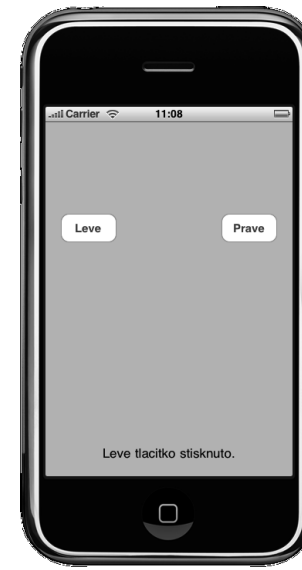
# KAPITOLA 3

# Základní interakce

Aplikace Hello World byla dobrým úvodem do vývoje v Cocoa Touch, ale scházela jí jedna zásadní vlastnost: schopnost interakce s uživatelem. Bez ní jsou možnosti vaší aplikace velmi omezené.

V této kapitole napíšete trochu komplexnější aplikaci, která bude mít dvě tlačítka a jeden popisek (viz obr. 3.1). Když uživatel poklepe na jedno z tlačítek, text popisku se změní. Možná se vám zdá, že to je příliš jednoduchý příklad, ale naučíte se na něm, jak používat další klíčové koncepty, které budete potřebovat k využití kontrolních prvků ve vašich aplikacích pro iPhone.




**Obrázek 3.1** Jednoduchá aplikace se dvěma tlačítky, kterou budete programovat v této kapitole

## Softwarová architektura Model-Vzhled-Řízení

Předtím, než se dáte do samotného návrhu aplikace, musíme probrat trochu teorie. Návrháři prostředí Cocoa Touch se při jeho psaní řídili konceptem Model-Vzhled-Řízení (Model-View-Controller, zkráceně MVC), který představuje velmi logický způsob rozdělení zdrojového kódu, který tvoří aplikaci založenou na grafickém uživatelském rozhraní. V sou-

časné době vycházejí téměř všechna objektově orientovaná prostředí a frameworky z konceptu MVC, ale pouze několik z nich se řídí touto koncepcí tak striktně, jako Cocoa Touch.

Model MVC rozděluje funkcionalitu aplikace do tří různých kategorií:

- *Model*: Třídy, které obsahují data vaší aplikace. Někdy je také model označován jako jádro aplikace.
- *Vzhled (View)*: Tvoří okna, ovládací prvky a další komponenty, které uživatel vidí a se kterými může interagovat.
- *Řízení (Controller)*: Spojuje model a vzhled a představuje logickou součást aplikace, která rozhoduje o tom, jak aplikace naloží se vstupem, který jí uživatel předá. Do češtiny je controller zpravidla překládán jako řízení (či nesprávně řadič).

**Poznámka**

Ačkoliv se pro překlad výrazu controller nabízí český kontrolér, není možné jej použít, jelikož vrstva řízení pouze nekontroluje (jak by z českého termínu vyplývalo), nýbrž především řídí chování aplikace. Asi nejlépe by funkci této vrstvy vystihovaly termíny dispečink a dispečer, které se ale v softwarové terminologii naneštěstí již používají v jiném významu. Rozhodli jsme se proto pro již poměrně zažitý překlad vrstva řízení (jako obecné označení v rámci softwarové architektury) a třídy řízení a řídící objekty (pro konkrétní jednotky, které tuto vrstvu implementují).

Cílem návrhu aplikace podle této architektury je maximálním možným způsobem od sebe odlišit tyto tři součásti jejího zdrojového kódu. Jakýkoliv objekt, který naprogramujete, by mělo být možné okamžitě zařadit do jedné z těchto tří kategorií a neměl by mít téměř a nebo vůbec žádnou funkcionalitu, která by spadala do některé z ostatních dvou kategorií. Objekt tlačítka by například neměl obsahovat zdrojový kód určený ke zpracování dat, když jej uživatel stiskne, a zdrojový kód, který pracuje s daty o bankovním účtu, by neměl zároveň na obrazovce zobrazovat tabulku s informacemi o prováděných finančních transakcích.

Architektura MVC pomáhá zajistit maximální opětovnou využitelnost naprogramovaných objektů. Třídu, která tvoří obecně použitelné tlačítko, lze použít v jakékoliv aplikaci. Třídu tvořící tlačítko, které po svém stisknutí provádí nějaký konkrétní výpočet, lze použít pouze v aplikaci, pro kterou byla původně napsána.

Při psaní aplikací v Cocoa Touch budete komponenty jejich vzhledu vytvářet primárně prostřednictvím programu Interface Builder, ačkoliv někdy budete muset upravit jejich zdrojový kód, nebo vytvořit podtřídu existujících rámců a ovládacích prvků.

Komponent model vytvoříte tak, že použijete třídy jazyku Objective-C navržené za účelem uchování dat vaší aplikace a nebo prostřednictvím nástroje Core Data, o kterém se dozvíte více v kapitole 11. Pro aplikaci, kterou budete programovat v této kapitole, žádné objekty modelu vytvářet nebudete, protože aplikace neuchovává žádná data – o objektech, které tvoří model, se dozvíte víc v následujících kapitolách, až budete programovat komplexnější aplikace.

Komponent řízení bude obvykle tvořen třídami, které sami vytvoříte a které budou specifické pro vaši aplikaci. Řízení mohou tvořit pouze vaše vlastní třídy (podtřídy třídy `NSObject`), ale častěji půjde o podtřídy třídy jednoho z obecných ovládacích prvků frameworku UIKit, jako je například třída `UIViewController`, o které se dozvíte více již za okamžik. Napíšete-li vaši třídu jako podtří-

du některé z těchto tříd, které již existují, získáte „zdarma“ mnoho funkcionality, kterou byste jinak museli sami programovat.

Až se s Cocoa Touch více seznámíte, bude vám jasné, že se třídy frameworku UIKit řídí pravidly konceptu MVC. Budete-li při návrhu vašich aplikací podvědomě stále myslet na to, že byste se měli tímto konceptem řídit, bude váš zdrojový kód čistší a bude se snáze udržovat.

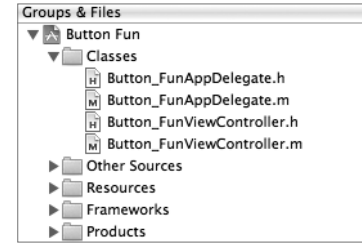
# Tvorba projektu

Nadešel čas vytvořit váš Xcode projekt. Použijte stejnou šablonu, kterou jste použili pro váš projekt v předchozí kapitole: *View-based Application*. Již brzy začnete používat i ostatní šablony, ale díky tomu, že nyní opět začnete s touto jednoduchou šablonu, budete moci lépe sledovat, jak spolu objekty vzhledu a řízení v aplikaci spolupracují. Jděte na to a vytvořte váš projekt a uložte jej jako *Button Fun* (Hrátky s tlačítky). Setkáte-li se při zakládání projektu s jakýmikoliv problémy, podívejte se do předchozích kapitol, kde najdete popis příslušných kroků.

Asi si vzpomínáte, že šablona projektu pro vás vytvořila nějaké třídy. Tyto třídy najdete i ve vašem novém projektu, ačkoliv některé z nich se budou jinak jmenovat, protože jejich názvy jsou založené na názvu projektu.

# Vytvoření řízení vzhledu

Za chvíli navrhnete vzhled (uživatelské rozhraní) vaší aplikace prostřednictvím nástroje Interface Builder stejně, jako jste to udělali v předchozí kapitole. Avšak předtím, než se do toho dáte, se podíváme na některé soubory se zdrojovým kódem, které jsou pro vás předpřipravené, a trochu je změníte. Ano, nyní konečně přijdete o své panenství a napíšete nějaký zdrojový kód.




**Obrázek 3.2** Soubory tříd, které pro vás vytvořila šablona projektu

Předtím, než tyto změny začnete provádět, se podívejte na předpřipravené soubory. V okně projektu rozbalte složku *Classes* – objeví se čtyři soubory, které obsahuje (viz obr. 3.2).

Tyto čtyři soubory tvoří dvě třídy – každá z nich obsahuje soubor s příponou *.m* a *.h*. Aplikace, kterou vytváříte, bude mít pouze jeden rámec, a třída řízení, která je zodpovědná za jeho správu, má název `Button_FunViewController`. Slova `Button_Fun` pocházejí z názvu projektu a `ViewController` znamená, že tato třída je řídícím objektem rámce vzhledu (tedy instance třídy UIView). Klepněte na soubor *Button_FunViewController.h* v tabuli *Groups & Files* a podívejte se na obsah souboru:

```
#import <UIKit/UIKit.h>

@interface Button_FunViewController : UIViewController {
}

@end
```

Moc toho není, že? Jedná se podtřídu třídy `UIViewController`, jedné z obecných tříd řízení, o kterých jsme se již zmínili. Jde o součást frameworku UIKit, která vám poskytuje velkou funkcionalitu,

aniž byste museli cokoliv sami udělat. Xcode sice neví, jaká bude přesně specifická funkcionalita vaší aplikace, ale ví, že aplikace nějakou funkcionalitu mít bude, a za tímto účelem tuto třídu vytvořil.

Podívejte se znovu na obrázek 3.1. Váš program se skládá ze dvou tlačítek a textového popisku, který reaguje na stisk jednoho či druhého tlačítka. Všechny tři tyto objekty vytvoříte v nástroji Interface Builder. Protože budete také psát nějaký zdrojový kód, musí existovat nějaký způsob, jakým bude váš zdrojový kód interagovat s objekty, které vytvoříte v nástroji Interface Builder, není-liž pravda?

Samozřejmě to tak je. Vaše třída řízení se může odkazovat na objekty v nib souboru prostřednictvím speciální instanční proměnné, které se říká **outlet**. Představujte si outlet jako ukazatel, který ukazuje na objekt v nib souboru. Představte si například, že jste vytvořili popisek v programu Interface Builder a chcete změnit jeho text prostřednictvím zdrojového kódu. Když outlet deklarujete a spojíte jej s objektem popisku, můžete outlet ve vašem kódu použít ke změně textu popisku. Za chvíli vám ukážeme, jak se to dělá.

### Poznámka

Termín outlet je do češtiny jen obtížně přeložitelný. Jelikož je navíc zcela specifický pro prostředí Cocoa a Cocoa Touch, rozhodli jsme se jej nepřekládat a raději jej, podobně jako framework, počeštit. Pro čtenáře tak bude snadnější jak práce v aplikaci Interface Builder, tak případné studování bohaté dokumentace firmy Apple.

Objekty uživatelského rozhraní uložené v nib souboru mohou na druhou stranu spouštět speciální metody v řídící třídě. Tyto speciální metody se nazývají **action methods** – **metody akcí** (zkráceně **akce**). Můžete například nástroji Interface Builder sdělit, že pokud uživatel uvolní stisk tlačítka (odtáhne svůj prst od obrazovky) na místě, kde je tlačítko, má se zavolat specifická metoda akce ve vašem zdrojovém kódu.

Jak jsme avizovali již na začátku této kapitoly, aplikace Button Fun bude mít dvě tlačítka a popisek.

Ve vašem zdrojovém kódu vytvoříte outlet, který bude na popisek ukazovat a který vám bude umožňovat měnit jeho text. Dále vytvoříme metodu s názvem `buttonPressed:`, která se spustí pokaždé, když uživatel klepne na jedno z tlačítek. Metoda `buttonPressed:` změní text popisku tak, aby bylo zřejmé, které tlačítko uživatel stisknul.

K vytvoření tlačítek a popisku použijete Interface Builder a pak s pomocí myši spojíte popisek s jeho outletem a tlačítka s akcí `buttonPressed:`.

Než však začnete programovat, projděte si ještě následující informace o outletech a akcích.

## Outlety

Outlety jsou instanční proměnné deklarované prostřednictvím klíčového slova `IBOutlet`. Deklarace outletu v hlavičkovém souboru vaší třídy řízení může vypadat takto:

```
@property (nonatomic, retain) IBOutlet UIButton *myButton;
```

Klíčové slovo `IBOutlet` se definuje takto:

```
#ifndef IBOutlet
#define IBOutlet
#endif
```

Co to znamená? Pokud jde o kompilátor, `IBOutlet` neznamená nic. Jediný účel tohoto klíčového slova spočívá v tom, že funguje jako informace pro Interface Builder, která říká, že se jedná o instanční proměnnou, kterou chcete připojit k nějakému objektu v nib souboru. Každá instanční proměnná, kterou vytvoříte a chcete připojit k nějakému objektu v nib souboru, musí být uvedena klíčovým slovem `IBOutlet`. Když Interface Builder otevřete, projde hlavičkové soubory vašich projektů, najde v nich toto slovo a umožní vám propojit váš zdrojový kód s objekty v nib souboru na základě těchto (a žádných jiných) proměnných. Za chvíli se naučíte, jak v nástroji Interface Builder vytvořit spojení mezi outletem a objektem uživatelského rozhraní.

**Změny v používání outletů**

V první verzi této knihy klíčové slovo `IBOutlet` vždy předcházelo deklaraci instanční proměnné:

```
IBOutlet UIButton *myButton;
```

Mezi tím však společnost Apple ve svých vzorových zdrojových kódech začala přesouvat klíčové slovo `IBOutlet` do deklarace atributů:

```
@property (nonatomic, retain) IBOutlet UIButton *myButton;
```

Použít můžete oba způsoby a většinou mezi nimi není ve výsledku žádný rozdíl – až na jednu výjimku. Deklarujete-li atribut s jiným názvem, než má instanční proměnná, která za ním stojí (což lze provést v direktivě `@synthesize`), musíte klíčové slovo `IBOutlet` umístit do deklarace atributu, chcete-li, aby správně fungovalo. Pokud máte potíže koncepci atributu pochopit, nebojte se, budeme se jí za chvíli podrobněji věnovat.

Ačkoliv lze použít oba výše uvedené postupy, řídíme se pravidlem firmy Apple a klíčové slovo `IBOutlet` umisťujeme vždy do deklarace atributu.

Více informací o nových vlastnostech atributů v Objective-C najdete v druhém vydání knihy *Objective-C on the Mac*, kterou napsali Mark Dalrymple a Scott Knaster (Apress 2008), a v manuálu *The Objective-C 2.0 Programming Language*, který je dostupný na webu společnosti Apple určeném pro vývojáře:

*http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf*

## Akce

Akce jsou metody, které jsou součástí třídy řízení. Při jejich deklaraci se také používá zvláštní klíčové slovo – `IBAction` – které říká nástroji Interface Builder, že tato metoda je akcí a lze ji spustit prostřednictvím ovládacího prvku. Deklarace metody akce obvykle vypadá takto:

```
- (IBAction)doSomething:(id)sender;
```

Konkrétní název metody může být jakýkoliv, ale typ jejího výsledku musí být `IBAction`, což je stejné, jako když deklarujete typ výsledku jako `void`. Jedná se o jiný způsob, jak sdělit, že metoda akce nevrací žádnou hodnotu. Vstupem metody akce je obvykle jeden parametr, který se obvykle definuje jako `id` a kterému se obvykle přiřadí název `sender`. Ovládací prvek, který akci spouští, použije parametr `sender` k tomu, aby předal odkaz na sebe sama. Když je tedy například volání vaší metody akce výsledkem stisknutí tlačítka, bude parametr `sender` obsahovat odkaz na specifické tlačítko, které uživatel stisknul.

Jak za chvíli uvidíte, váš program bude parametr `sender` používat pro nastavení textu popisku na „Leve“ nebo „Prave“ v závislosti na tom, které tlačítko uživatel stisknul. Pokud nepotřebujete vědět,

který ovládací prvek metodu zavolal, můžete také akční metody definovat bez parametru `sender`. Taková deklarace by vypadala takto:

```
- (IBAction)doSomething;
```

Ale nic se nestane, když metodu akce deklarujete s parametrem `sender`, a pak jej nevyužíváte. Pravděpodobně se setkáte se spoustou zdrojového kódu, který takto funguje, protože dříve metody akce v Cocoa musely akceptovat parametr `sender`, ať už jej používaly, či nikoliv.

## Přidávání akcí a outletů do třídy řízení vzhledu

Když tedy nyní víte, co to jsou outlety a akce, pojďte jeden outlet a jednu akci přidat do třídy řízení. Abyste mohli měnit text popisku, budete potřebovat outlet. Pro tlačítka nebudete outlet potřebovat, protože je nebudete nijak měnit.

Deklarujete také jedinou metodu akce, kterou budou volat obě tlačítka. I když má mnoho ovládacích prvků své vlastní metody akcí, můžete také použít jednu metodu akce ke zpracování vstupu z více ovládacích prvků, což uděláte i v tomto případě. Vaše akce zjistí název tlačítka z jeho parametru sender a použije zásuvku popisku ke nastavení jeho textu podle stisknutého tlačítka. Za chvíli vám ukážeme, jak se to dělá.

### Poznámka

Protože Xcode pro vás vytvoří soubory, které již obsahují část zdrojového kódu, který budete potřebovat, budete často doplňovat zdrojový kód do již existujícího souboru. Když v knize narazíte na výpis zdrojového kódu, jako následující kód ze souboru *Button_FunViewController.h*, je vždy jeho část, která je zapsána `normálním` fontě již v souboru předpřipravená. `Tučným` fontem je pak vytištěn zdrojový kód, který musíte do souboru přidat.

Jděte na to a přidejte do souboru *Button_FunViewController.h* následující zdrojový kód:

```
#import <UIKit/UIKit.h>
@interface Button_FunViewController : UIViewController {
    UILabel *statusText;
}
@property (nonatomic, retain) IBOutlet UILabel *statusText;
- (IBAction)buttonPressed:(id)sender;
@end
```

Pokud jste již s jazykem Objective-C 2.0 někdy pracovali, pravděpodobně již znáte deklaraci `@property`; pokud ne, možná vám tento řádek přijde poněkud zvláštní. Ale nebojte se, koncepce atributů je v Objective-C opravdu jednoduchá. Pojďte se na ni v rychlosti podívat, protože atributy jsou v Objective-C relativně nové a budete je v této knize velmi často používat. Dokonce i když jste již mistr v jejich používání, čtěte prosím dále, protože se dozvíte některé informace specifické pro Cocoa Touch, které se vám budou hodit.

#### Atributy v Objective-C

Předtím, než byly atributy do Objective-C přidány, programátoři tradičně definovali páry metod, které nastavovaly a navracely hodnoty pro každou instanční proměnnou ve třídě. Tyto metody se

souhrnně nazývají **přístupové metody** (accessors) a jsou dvě – **návratová** (anglicky **getter** či **accessor**) a **nastavovací** (**setter** či **mutator**) a mohou vypadat například nějak takto:

```
- (id) foo {
    return foo;
}
- (void) setFoo: (id) aFoo {
    if (aFoo != foo) {
        [aFoo retain];
        [foo release];
        foo = aFoo;
    }
}
```

Ačkoliv tento přístup je stále plnohodnotný, deklarace `@property` vám nyní umožňuje dát tomuto únavnému procesu vytváření návratových a nastavovacích metod sbohem. Deklarace `@property`, kterou jste právě napsali, spolu s dalšími deklaracemi v implementačním souboru (`@synthesize`), které vám za chvíli ukážeme, sdělí kompilátoru, aby návratové a nastavovací metody vytvořil při kompilaci. Stále musíte deklarovat instanční proměnné, které za nimi stojí, ale již nemusíte definovat návratové ani nastavovací metody.

V této deklaraci následují za klíčovým slovem `@property` volitelné parametry uzavřené v kulatých závorkách. Tyto parametry dále definují, jakým způsobem kompilátor vytvoří návratové a nastavovací metody. Volitelné parametry použité v naší deklaraci se budou při definovaní atributů v aplikacích pro iPhone objevovat často:

```
@property (nonatomic, retain) UILabel *statusText;
```

První z obou volitelných parametrů, `retain`, říká kompilátoru, aby zaslal jakémukoliv objektu, kterému tento atribut přidělíte, zprávu, že má přidržet (retain) instanční proměnnou. Instanční proměnná, která stojí za vaším atributem, tak nebude odstraněna z paměti, dokud ji budete potřebovat. To je důležité, protože výchozí chování (`assign`) je navrženo pro potřeby spolupráce s automatickou správou paměti, což je vlastnost Objective-C, která není pro iPhone v současné době k dispozici. Takže když definuje nějaký atribut, který je objektem (protikladem takového atributu je například jednoduchý datový typ `int`), měli byste v seznamu volitelných parametrů uvést `retain`. Když deklarujete atribut pro `int`, `float` a nebo jiný jednoduchý datový typ, nemusíte volitelné parametry uvádět.

Druhý z volitelných parametrů, `nonatomic`, mění způsob, jakým se generují návratové a nastavovací metody. Abychom příliš nezacházeli do podrobností, řekněme, že ve výchozím nastavení se tyto metody vytváří s dodatečným zdrojovým kódem, který se hodí při psaní programů s více vlákny. Tento dodatečný zdrojový kód, i když ho není mnoho, je však zbytečný, když deklarujete ukazatel na objekt uživatelského rozhraní, takže když použijete volitelný parametr `nonatomic`, ušetříte tak trochu zbytečné práce. Někdy bude potřeba tento parametr vynechat, ale obecně vzato budete při psaní aplikací pro iPhone tento parametr většinou používat.

Objective-C 2.0 má další příjemnou vlastnost, kterou budete využívat spolu s atributy. V této verzi jazyku je možné používat **tečkovou notaci**. Tradičně, když jste chtěli využít návratovou metodu, poslali jste objektu zprávu podobnou této:

```
myVar = [someObject foo];
```

Tento přístup je stále plně využitelný. Ale když definujete atribut, můžete také použít tečkovou notaci, podobně jako v jazycích Java, C++ a C#:

```
myVar = someObject.foo;
```

Z pohledu kompilátoru jsou tyto dva příkazy úplně identické; používejte ten z nich, který je pro vás přirozenější. Tečkovou notaci lze využít i v případě nastavovacích metod. Následující příkaz:

```
someObject.foo = myVar;
```

má identickou funkci jako tento příkaz:

```
[someObject setFoo:myVar];
```

#### Deklarace metody akce

Po deklaraci atributu jste do zdrojového kódu přidali další řádek:

```
- (IBAction)buttonPressed:(id)sender;
```

Toto je deklarace vaší metody akce. Když sem deklaraci umístíte, informujete ostatní třídy a Interface Builder o tom, že vaše třída má metodu akce s názvem `buttonPressed:`.

## Přidávání akcí a outletů do implementačního souboru

Práce na hlavičkovém souboru řídící třídy je prozatím hotová, takže jej uložte a potom jednou klepněte na implementační soubor třídy – *Button_FunViewController.m*. Soubor by měl mít tento obsah:

```
#import "Button_FunViewController.h"
@implementation Button_FunViewController
/*
// Vyhrazený inicializátor. Změňte, chcete-li provést nastavení,
// která je třeba udělat před načtením rámce.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:
(NSBundle *)nibBundleOrNil {
if (self=[super initWithNibName:nibNameOrNil bundle:nibBundleOrNil]) {
// Inicializace specifická pro objekty této třídy.
}
return self;
}
*/
/*
// Implementujte loadView, chcete-li vytvořit hierarchii rámce programově,
// bez použití nib souboru.
- (void)loadView {
}
*/
/*
// Implementujte viewDidLoad, chcete-li po načtení rámce provést dodatečná
// nastavení, obvykle z nib souboru.
- (void)viewDidLoad {
[super viewDidLoad];
}
*/
/*
// Změňte, chcete-li umožnit jinou orientaci rámce než výchozí svislou.
- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation {
// Vrací YES pro podporované orientace.
return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
```

```
*/
- (void)didReceiveMemoryWarning {
// Uvolní rámec vzhledu, pokud nemá nadřazený rámec.
[super didReceiveMemoryWarning];
// Uvolněte data z vyrovnávací paměti, nepoužívané obrázky ap.
}
- (void)viewDidUnload {
// Uvolněte jakékoliv přidržené rámce vnořené do rámce hlavního.
// například self.myOutlet = nil;
}
- (void)dealloc {
[super dealloc];
}
@end
```

### Poznámka

Komentáře v nově vytvořených souborech budou samozřejmě anglicky, ale my jsme vám je pro začátek přeložili, abyste se mohli soustředit na samotný kód a ne na překlad komentářů.

Společnost Apple sem předpřipravila některé metody, jejichž implementaci pravděpodobně změníte neboli překryjete (override), a zahrnula jejich základy do implementačního souboru. Některé z nich jsou označené jako komentáře a lze je podle potřeby buď opět zařadit do proveditelné sekce, nebo vymazat. Metody, které jako komentáře označené nejsou, využívá buď šablona, nebo se obecně využívají tak často, že sem byly zařazeny proto, abyste ušetřili čas potřebný k jejich napsání. Pro tuto aplikaci nebudete potřebovat žádnou z metod označených jako komentář, takže je pojďte smazat, což zkrátí zdrojový kód a zpřehlední vkládání kódu nového.

### Poznámka

Anglický temín override je velmi často chybně překládán jako potlačení, s čímž jste se již pravděpodobně setkali. Jeho skutečný význam je ale zcela jiný, metody, které „overridnete“, rozhodně nijak nepotlačujete, prostě a jednoduše změníte jejich implementaci, čímž **překryjete** původní implementaci v rodičovské třídě.

Když jsou tedy tyto metody smazané, přidejte do souboru následující zdrojový kód. Až to budete mít, řekneme vám, co jste vlastně udělali:

```
#import "Button_FunViewController.h"

@implementation Button_FunViewController
@synthesize statusText;

- (IBAction)buttonPressed:(id)sender {
    NSString *title = [sender titleForState:UIControlStateNormal];
    NSString *newText = [[NSString alloc] initWithFormat:
                         @"%@ tlacitko stisknuto.", title];
    statusText.text = newText;
    [newText release];
}
- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning]; // Uvolní rámec vzhledu, pokud
```

```
    // nemá nadřazený rámec.
    // Uvolněte data z vyrovnávací paměti, nepoužívané obrázky ap.
}
- (void)viewDidUnload {
    // Uvolněte jakékoliv přidržené rámce vnořené do rámce hlavního.
    // například self.myOutlet = nil;
    self.statusText = nil;
}
- (void)dealloc {
    [statusText release];
    [super dealloc];
}

@end
```

Hotovo? Tak se pojďte podívat na nově přidaný zdrojový kód. Nejprve jste do kódu přidali tento řádek:

```
@synthesize statusText;
```

Tento řádek říká kompilátoru, aby automaticky vytvořil návratovou a nastavovací metodu. Tento řádek vytvoří ve vaší třídě dvě „neviditelné" metody: `statusText` a `setStatusText:`. Sice jste je sami nenapsali, ale přesto jsou připravené k použití.

Další nová součást zdrojového kódu představuje implementaci vaší metody akce, která se bude volat, když uživatel stiskne jedno z tlačítek:

```
- (IBAction)buttonPressed:(id)sender {
    NSString *title = [sender titleForState:UIControlStateNormal];
    NSString *newText = [[NSString alloc] initWithFormat:
                         @"%@ tlacitko stisknuto.", title];
    statusText.text = newText;
    [newText release];
```

Pamatujte, že metodě akce se jako parametr předá ovládací prvek nebo objekt, který ji zavolal. Takže ve vaší aplikaci bude parametr `sender` vždy odkazovat na stisknuté tlačítko. To je velmi šikovný mechanismus, který umožňuje, aby jedna metoda akce obsluhovala vstup z více ovládacích prvků – což je přesně to, co zde děláte: obě tlačítka volají tuto metodu, a které z nich to je, určuje hodnota parametru `sender`. První řádek této metody vytáhne z parametru `sender` název tlačítka:

```
NSString *title = [sender titleForState:UIControlStateNormal];
```

### Poznámka

Když požadujete informaci o názvu tlačítka, musíte zadat jeden z jeho možných stavů. Možné stavy ovládacího prvku jsou čtyři: normal, což znamená, že je ovládací prvek aktivní, ale zrovna se nepoužívá; highlighted, což znamená, že je ovládací prvek stisknut, nebo jinak aktivován; disabled, což znamená, že tento ovládací prvek není aktivní a nelze jej použít; a selected, což je stav, ve kterém se mohou nacházet pouze některé ovládací prvky a který znamená, že je ovládací prvek vybraný. `UIControlStateNormal` reprezentuje normální stav ovládacího prvku a jde o stav, ve kterém budete ovládací prvky většinou používat. Když hodnoty pro ostatní stavy neuvedete, budou mít stejnou hodnotu jako stav normal.

Poté vytvoříte nový řetězec založený na názvu tlačítka:

```
NSString *newText = [[NSString alloc] initWithFormat:
                     @"%@ tlacitko stisknuto.", title];
```

Tento nový řetězec bude obsahovat název tlačítka a text „tlacitko stisknuto.“ Takže když stisknete tlačítko s názvem „Leve“, bude tento řetězec obsahovat text „Leve tlacitko stisknuto.“.

### Poznámka

Ačkoliv operační systém i jazyk Objective-C plně podporují znakovou sadu Unicode, při zadávání textu do statických řetězců musíte vystačit se standardní ASCII sadou, která nepodporuje diakritiku. Aby se nám nemíchal text s a bez diakritiky, necháme i názvy tlačítek a dalších prvků uživatelského rozhraní bez ní. Stejně tak budeme postupovat i v dalších příkladech a kapitolách. Jak zadávat ve zdrojovém kódu text s diakritikou, se dozvíte v kapitole 17, která pojednává o lokalizaci aplikací. Nebojte se, jak uvidíte, je to velmi jednoduché, jelikož návrháři prostředí NeXTSTEP již v osmdesátých letech prozíravě počítali s možností aplikací v jiných jazycích než v angličtině.

Dále nastavíte jako text popisku tento nový řetězec:

```
statusText.text = newText;
```

V tomto případě se používá tečková notace, ale příkaz je možné zapsat také takto: `[statusText setText:newText];`. Nakonec uvolníte řetězec:

```
[newText release];
```

Nutnost uvolnění objektů poté, co jste s nimi hotoví, nelze podceňovat. iPhone je zařízení s velmi omezenými prostředky a dokonce i malé úniky paměti mohou způsobit zhroucení vaší aplikace. Měli bychom také zdůraznit, že jste *neudělali* toto:

```
NSString *newText = [NSString stringWithFormat:
                     @"%@ tlacitko stisknuto.", title];
```

Tento zdrojový kód by fungoval úplně stejně jako ten, který jste použili. Metody tříd, jako je tato, se nazývají **konvenční** metody (**convenience** nebo **factory** methods). Tyto metody vrací automaticky uvolněný objekt. Řídíte-li se obecným pravidlem pro využívání paměti, které říká, že „pokud jste nějakou paměť nealokovali nebo nepřidržujete její obsah, neměli byste jí ani uvolňovat“, nemusíte tyto objekty, které se z paměti uvolňují automaticky, uvolňovat manuálně, aniž byste je záměrně v paměti uchovávali, a výsledkem jejich využití je často kratší a čitelnější zdrojový kód.

Avšak použití těchto konvenčních metod znamená zvýšení nákladů, protože využívají automatický zásobník pro automatické uvolňování paměti. Paměť alokovaná pro automaticky uvolňované objekty zůstane alokovaná ještě nějakou dobu poté, co již s těmito objekty nepracujete. V případě aplikací pro systém Mac OS X, který využívá swap soubory a má k dispozici relativně mnoho fyzické paměti, jsou náklady vynaložené na automatické uvolňování objektů zanedbatelné, ale v případě iPhonu, mohou mít tyto objekty škodlivý vliv na způsob, jakým vaše aplikace paměť používá. Automatický zásobník se vyplatí používat, ale pouze pokud to opravdu potřebujete a ne pouze proto, abyste ušetřili jeden nebo dva řádky zdrojového kódu.

### Poznámka

Pozor, nezaměňujte automatickou správu paměti (garbage collection), která v iPhone OS není k dispozici, s automaticky uvolňovanou pamětí neboli automatickým zásobníkem (autorelease pool), který k dispozici je. Podrobnosti o obou technologiích se dozvíte například v dokumentech *Memory Management Programming Guide for Cocoa* a *Garbage Collection Programming Guide*, které naleznete na vývojářských stránkách firmy Apple.

Další novou součástí, kterou jste do programu přidali, je tento řádek:

```
self.statusText = nil;
```

Tohoto řádku si zatím moc nevšímejte, k čemu je potřeba, vám vysvětlíme v následující kapitole. Prozatím si jenom zapamatujte, že musíte nastavit všechny outlety ve vaší třídě v metodě `viewDidUnload` na `nil`.

### Tip

Pokud vás správa paměti v Obective-C trochu mate, měli byste si projít první z výše uvedených dokumentů. Dokonce i malé množství úniků paměti může v aplikaci pro iPhone způsobit velkou spoušť.

Poslední věcí, kterou jste udělali, bylo uvolnění outletu v metodě `dealloc`:

```
[statusText release];
```

Uvolnění této položky vám může připadat poněkud zvláštní. Možná si myslíte, že když jste ji nevytvořili, neměli byste být odpovědní za její uvolnění. Pokud jste již dříve pracovali se staršími verzemi Cocoa a Objective-C, máte možná pocit, že to, co děláte, je úplně špatně. Avšak protože jste pro každý z outletů implementovali atributy a použili jako jeden z jejich volitelných parametrů `retain`, je třeba outlet uvolnit. Interface Builder použije při přiřazování outletů vaší vygenerovanou nastavovací metodu a tato metoda bude přidržovat objekt, který je jí přiřazen, takže je důležité outlet uvolnit, abyste zabránili únikům paměti.

Předtím, než budete pokračovat, soubor nezapomeňte uložit. Až jej uložíte, stiskněte ⌘+B a zkompilujte jej, abyste se mohli ujistit, že jste při psaní neudělali žádnou chybu. V případě, že se soubor nezkompiluje, se vraťte zpět a porovnejte zdrojový kód v knize s vaším vlastním.

#### Vnořování zpráv

Někteří vývojáři často v Objective-C vnořují zprávy. Například byste se mohli setkat s podobným zdrojovým kódem, jako je tento:

```
statusText.text = [NSString stringWithFormat:@"%@ button pressed.",
    [sender titleForState:UIControlStateNormal]];
```

Tento jediný řádek zdrojového kódu funguje úplně stejně, jako čtyři řádky, které tvoří vaši metodu `button-Pressed:`. Kvůli čitelnosti nebudeme s výjimkou volání zpráv `alloc` a `init`, které jsou podle tradiční konvence téměř vždy vnořené, v této knize v drtivé většině případů zprávy v příkladech zdrojových kódů vnořovat.

# Delegát aplikace

Další dva soubory ve složce Classes implementují **delegáta vaší aplikace**. Cocoa Touch využívá **delegáty** velmi často. **Delegáti** jsou třídy zodpovědné za provádění určitých operací pro jiné objekty. Delegát aplikace vám umožňuje provádět určité operace v konkrétních předefinovaných okamžicích pro třídu `UIApplication`. Každá aplikace pro iPhone má jednu jedinou instanci třídy `UIApplication`, která je zodpovědná za její chod a zajišťuje její funkcionalitu, jako je například směrování vstupu na příslušnou třídu příslušného ovládacího prvku.

Třída `UIApplication` je standardní součástí frameworku UIKit a většinou pracuje skrytě, takže se jí obvykle nemusíte zabývat. V určitých přesně definovaných okamžicích však třída `UIApplication` volá specifické metody delegáta, pokud je delegát k dispozici, a disponuje příslušnou metodou. Pokud například potřebujete použít zdrojový kód, který se provede těsně předtím, než se aplikace ukončí, implementujete metodu `applicationWillTerminate:` do delegáta vaší aplikace a umístíte do ní tento kód. Tento způsob delegování umožňuje vaší aplikaci implementovat běžné způsoby chování, aniž byste museli použít podtřídu třídy `UIApplication` a nebo dokonce museli něco vědět o tom, jak tato třída uvnitř funguje.

Klepněte na *Button_FunAppDelegate.h* v tabuli *Groups & Files* a podívejte se na obsah hlavičkového souboru delegáta aplikace. Obsah souboru by měl vypadat nějak takto:

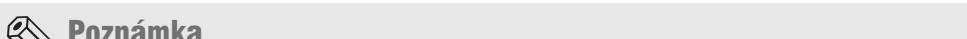
```
#import <UIKit/UIKit.h>

@class Button_FunViewController;

@interface Button_FunAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    Button_FunViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet Button_FunViewController
    *viewController;

@end
```

Tento soubor nemusíte nijak měnit a po implementaci vaší třídy řízení by vám měla být většina jeho obsahu povědomá. O jednom řádku jeho zdrojového kódu je však vhodné se zmínit podrobněji:

```
@interface Button_FunAppDelegate : NSObject <UIApplicationDelegate> {
```

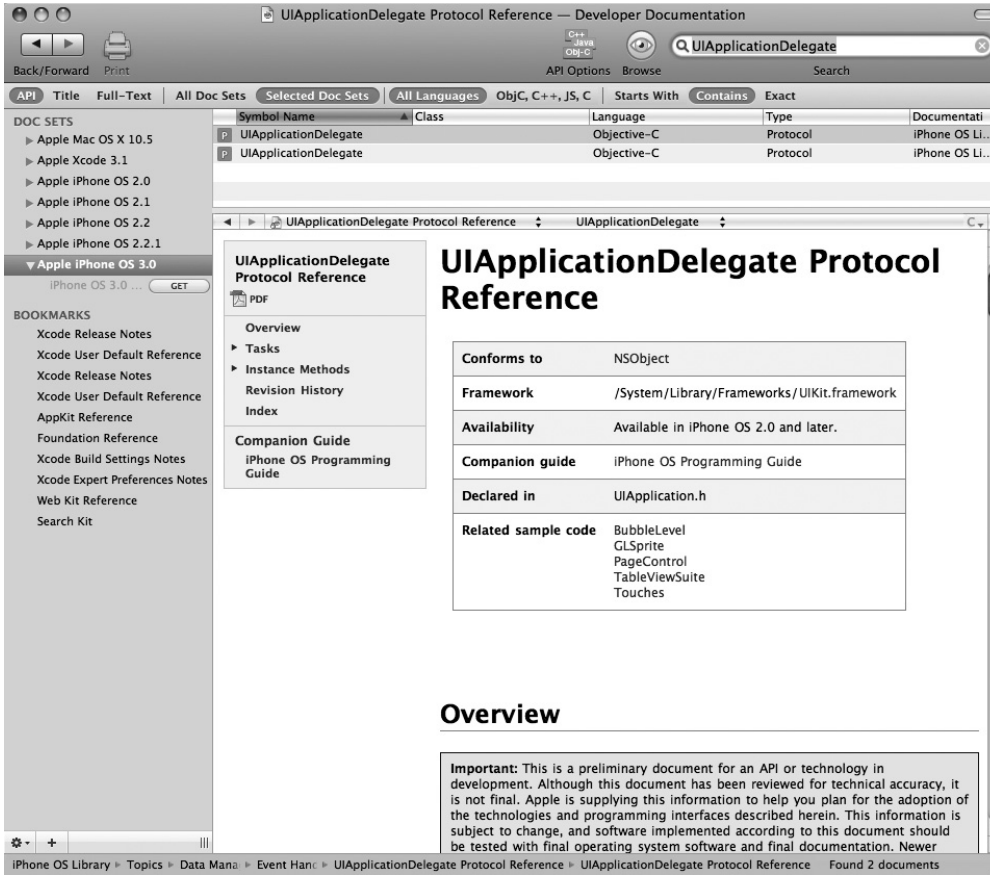
Vidíte hodnotu uvedenou mezi ostrými závorkami? To znamená, že tato třída odpovídá protokolu s názvem `UIApplicationDelegate`. Stiskněte klávesu Option a najeďte kurzorem na slovo `UIApplicationDelegate`. Váš kurzor by se měl změnit na zaměřovač; jakmile se kurzor změní, poklepejte myší. Otevře se program pro procházení dokumentací a zobrazí dokumentaci protokolu `UIApplicationDelegate` (viz obr. 3.3). Stejný trik funguje s třídami, protokoly a názvy kategorií, stejně jako s názvy metod, které se zobrazují v tabuli editoru. Jednoduše stiskněte klávesu Option a poklepejte na vybrané slovo; dokumentace pro toto slovo se otevře v okně pro procházení dokumentací.

Určitě se hodí naučit se jednoduše zobrazit dokumentaci, ale ještě důležitější je samotná dokumentace pro tento protokol. Najdete v ní seznam metod, které lze v delegátu aplikace implementovat, a kdy se tyto metody volají. Pravděpodobně stojí za to, abyste si jejich popis přečetli.



## Poznámka

Pokud jste již dříve pracovali s Objective-C, ale s verzí 2.0 pracujete poprvé, měli byste vědět, že protokoly nyní mohou specifikovat volitelné metody. Protokol `UIApplicationDelegate` obsahuje mnoho volitelných metod, které nemusíte v delegátu aplikace implementovat, pokud k tomu nemáte důvod.




**Obrázek 3.3** Dokumentace protokolu UIApplicationDelegate otevřená v programu pro procházení dokumentací

Klepněte na *Button_FunAppDelegate.m* a podívejte se na implementaci delegáta aplikace. Měla by vypadat takto:

```
#import "Button_FunAppDelegate.h"
#import "Button_FunViewController.h"

@implementation Button_FunAppDelegate

@synthesize window;
@synthesize viewController;
```

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // Na tomto místě můžete nastavit aplikaci po jejím spuštění
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}

- (void)dealloc {
    [viewController release];
    [window release];
    [super dealloc];
}

@end
```

Přímo uprostřed souboru vidíte, že delegát vaší aplikace implementuje jednu z metod protokolu: metodu `applicationDidFinishLaunching:`, která se, jak asi uhádnete, zavolá hned po tom, co aplikace dokončí všechnu práci s nastavením a je připravená začít interagovat s uživatelem.

Verze metody `applicationDidFinishLaunching:` ve vašem delegátu přidává rámec řízený třídou `Button_FunViewController.h` jako vnořený rámec do hlavního okna aplikace a činí jej viditelným v podobě, ve které bude vámi navržený rámec vidět uživatel. Aby tomu tak bylo, nemusíte sami nic dělat; vše zajistí zdrojový kód, který se vygeneroval jako součást šablony, kterou jste použili pro tento projekt.

Pouze jsme vám chtěli ukázat některé vlastnosti delegátů aplikací a jak vše dohromady funguje.

# Úprava souboru MainWindow.xib

Zatím jsme se podívali na čtyři soubory ve složce *Classes* vašeho projektu (dva soubory s příponou *.m* a dva s příponou *.h*). V předchozích kapitolách jste již pracovali se dvěma ze tří souborů ve složce *Resources*. Viděli jste ekvivalent souboru *Button_Fun-Info.plist*, když jste přidávali do vašeho projektu ikonu, a ekvivalent souboru *Button_FunViewController.xib*, když jste přidávali do aplikace váš popisek „Ahoj svete!“.

Ve složce *Resources* je ještě jeden soubor, o kterém budeme mluvit nyní. Soubor *MainWindow.xib* má na svědomí, že se delegát vaší aplikace, její hlavní okno a instance třídy řídící zobrazovaný rámec vytvoří za běhu programu. Pamatujte, že tento soubor je dodáván jako součást šablony projektu. Nemusíte v něm nic dělat ani měnit. Jedná se pouze o možnost sledovat, jaké operace probíhají na pozadí tvorby aplikace, abyste si mohli udělat lepší celkový obrázek.


**Obrázek 3.4** Okno MainWindow.xib vaší aplikace v nástroji Interface Builder

Rozbalte složku *Resources* v tabuli *Groups & Files* prostředí Xcode a poklepejte na soubor *MainWindow.xib*. Jakmile se otevře Interface Builder, podívejte se na hlavní okno nib souboru – okno označené jako *MainWindow.xib*, které by mělo vypadat stejně jako na obrázku 3.4.

První dvě ikony v tomto okně byste měli poznat, již jsme o nich mluvili v druhé kapitole. Vzpomeňte si také, že všechny ostatní ikony reprezentují objekty, které se vytvoří, když se nib soubor načte. Pojďte se podívat, co představuje druhá, třetí a čtvrtá ikona v okně.

**Poznámka**

Dlouhé názvy se ve výchozím rámci hlavního okna nib souboru zobrazují oříznuté (viz obr. 3.4). Když na chvíli zastavíte kurzor nad ikonou v okně, objeví se okénko s úplným názvem souboru. Všimněte si také, že názvy zobrazené v hlavním okně nemusí nutně odpovídat třídám, které za objekty stojí. Výchozí název nové instance bude obvykle nějakým způsobem odkazovat na třídu, která za touto instancí stojí, avšak názvy instancí lze měnit a často se tak i děje.

Třetí ikona reprezentuje instanci delegáta `Button_FunAppDelegate`. Čtvrtá ikona reprezentuje instanci třídy řízení `Button_FunViewController`. A konečně pátá ikona reprezentuje jedno jediné okno vaší aplikace (instanci třídy `UIWindow`). Tyto tři ikony indikují, že když se nib soubor načte, bude mít vaše aplikace jednu instanci delegáta – `Button_FunAppDelegate`; jednu instanci třídy řídící rámec vzhledu – `Button_FunViewController`; a jednu instanci třídy `UIWindow` (třídy, která reprezentuje jediné okno aplikace). Jak vidíte, Interface Builder toho umí mnohem víc, než jen vytvářet prvky uživatelského rozhraní. Umožňuje vám vytvářet také instance jiných tříd. To je neuvěřitelně výkonná vlastnost. Každý řádek zdrojového kódu, který nenapíšete, je řádek, pro který nemusíte provádět ladění a nemusíte ho udržovat. Zde vytváříte tři instance objektů při spuštění aplikace, aniž byste museli napsat jediný řádek zdrojového kódu.

Dobře, to je vše, co jsme vám zde chtěli ukázat, přátelé, pojďme dál. Ujistěte se, že jste tento nib soubor zavřeli. A pokud budete požádáni o jeho uložení, až jej budete zavírat, zvolte „No“, protože jste v něm neměli nic měnit.

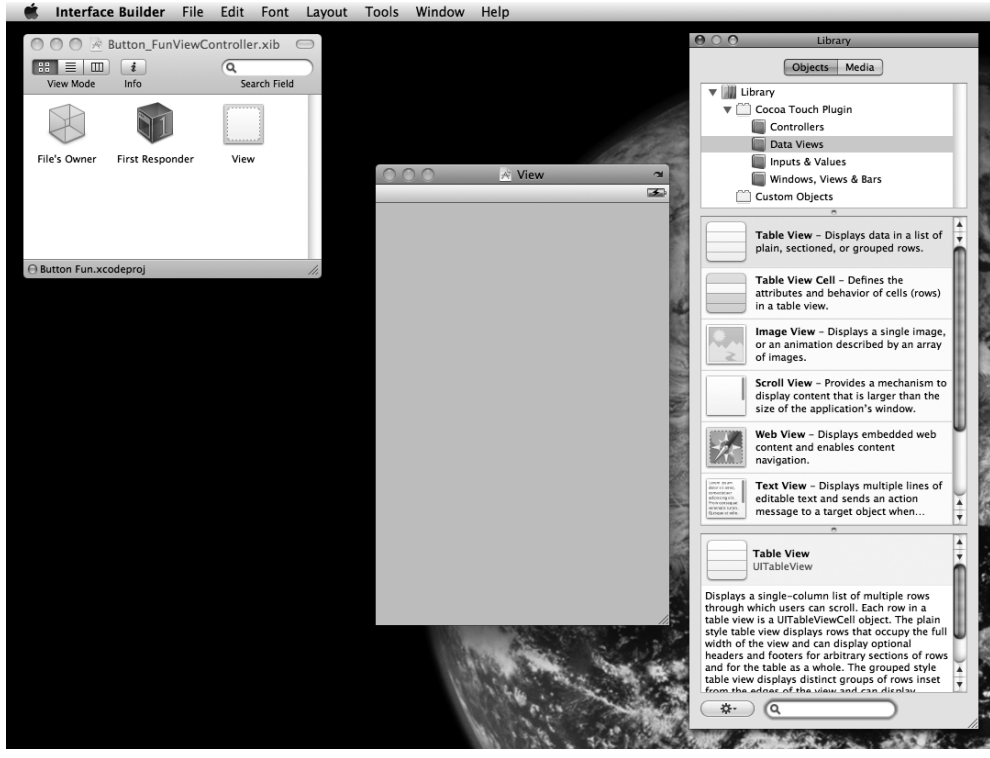
# Úprava souboru Button_FunViewController.xib

Nyní, když víte, které soubory tvoří váš projekt a jaké koncepty je drží pohromadě, obraťte svou pozornost k nástroji Interface Builder a procesu konstrukce vašeho uživatelského rozhraní.

## Vytvoření vzhledu v nástroji Interface Builder

V prostředí Xcode poklepejte na soubor *Button_FunViewController.xib* v tabuli *Groups & Files*. Nib soubor by se měl otevřít v programu Interface Builder. Ujistěte se, že se zobrazí knihovna. Pokud ne, zobrazíte ji tak, že zvolíte v menu **Tools** položku **Library**. Ujistěte se také, že je otevřené okno *View* nib souboru. Pokud ne, poklepejte na ikonu s názvem *View* v jeho hlavním okně (viz obr. 3.5).

Nyní jste připravení začít navrhovat vaše uživatelské rozhraní. Přetáhněte popisek (label) z knihovny do okna zobrazujícího rámec, stejně jako jste to udělali v předchozí kapitole. Umístěte popisek do levého dolního rohu rámce tak, aby se dotýkal levé a spodní modré pomocné čáry (viz obr. 3.6). Potom jej roztáhněte tak, aby se pravá strana popisku dotýkala pravé pomocné čáry.



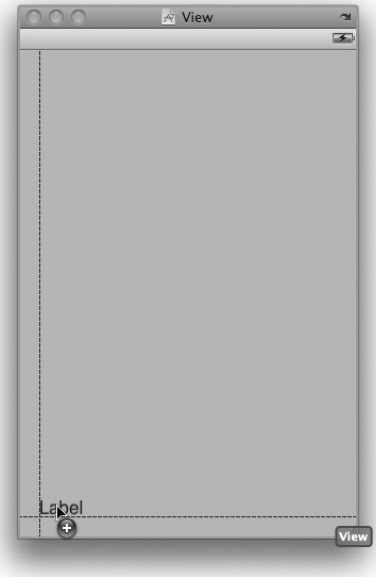

**Obrázek 3.5** Soubor Button_FunViewController.xib otevřený v programu Interface Builder



## Poznámka

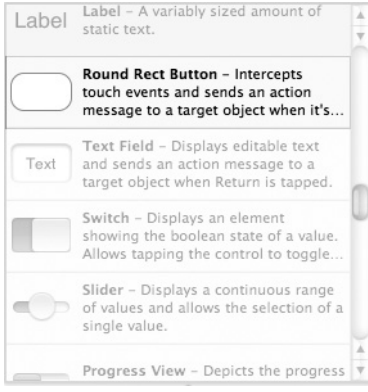
Tenké modré pomocné čáry vám pomáhají postupovat při návrhu uživatelského rozhraní v souladu se souborem pravidel *Apple Human Interface Guidelines* (kterému se obvykle zkráceně říká „HIG“). Ano, stejně jako v případě systému Mac OS X společnost Apple dodává soubor pravidel *iPhone Human Interface Guidelines* pro návrh aplikací pro iPhone. Soubor pravidel HIG vám říká, jak byste měli (a neměli) navrhovat uživatelské rozhraní. Opravdu byste si měli tato pravidla přečíst, protože se jedná o hodnotné informace, které by měl znát každý návrhář aplikací pro iPhone. Najdete je na adrese *http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/MobileHIG/*.

Až popisek umístíte, klepněte na něj a stiskněte ⌘+1 – otevře se inspektor. Zarovnejte text popisku na střed tak, že klepnete na příslušné tlačítko pro zarovnání textu (viz obr. 3.7).



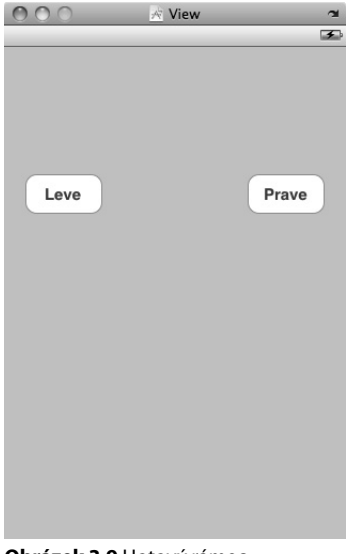

**Obrázek 3.6** Použití pomocných čar při umisťování objektů

Nyní poklepejte na popisek a smažte jeho text – dokud uživatel nestiskne jedno z tlačítek, neměl by mít popisek žádný text.

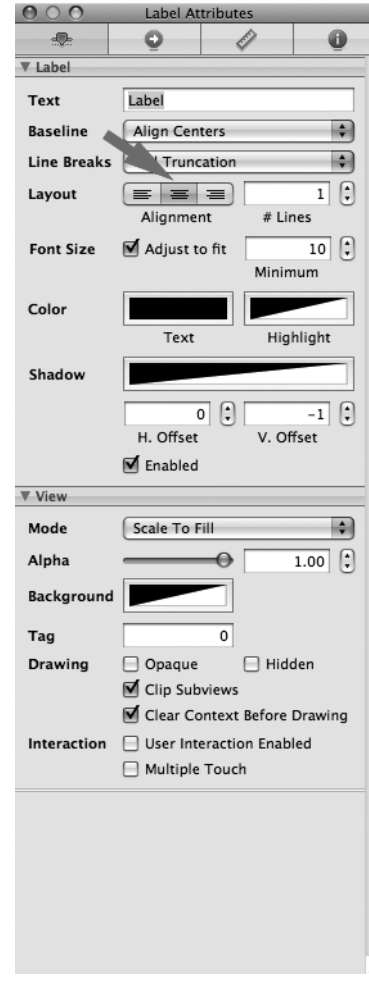



**Obrázek 3.8** Tlačítko Round Rect Button v knihovně

Potom přetáhněte z knihovny do rámce dvě tlačítka s názvem *Round Rect Button* (viz obr. 3.8).




**Obrázek 3.9** Hotový rámec




**Obrázek 3.7** Tlačítka pro zarovnání textu v inspektoru

Umístěte tlačítka vedle sebe, zhruba uprostřed rámce. Přesné umístění nehraje roli. Poklepejte na tlačítko vlevo. Budete moci upravit text tlačítka. Dejte se do toho na nastavte jeho text a „Leve“. Potom poklepejte na pravé tlačítko a změňte jeho text na „Prave“. Výsledek by měl vypadat jako náš rámec na obrázku 3.9.

## Propojení všeho dohromady

Nyní jsou všechny součásti vašeho uživatelského rozhraní hotové. Zbývá pouze všechny propojit, aby mohly spolupracovat.

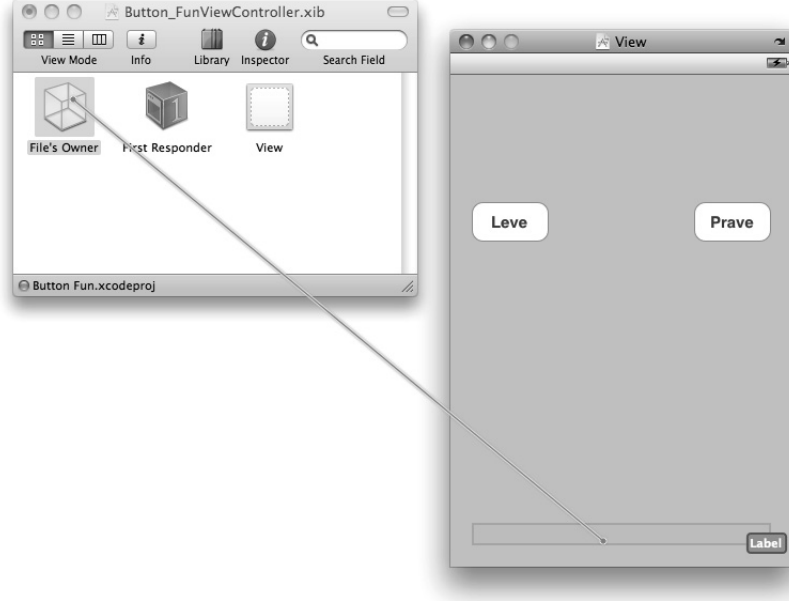
Prvním krokem je propojit objekt *File's Owner* a popisek v okně *View*. Proč *File's Owner*?

Když se instance třídy `UIViewController` nebo některé z jejích podtříd vytvoří, lze jí říci, aby se inicializovala ze souboru nib. V šabloně, kterou jste použili, se třída `Button_FunViewController` načte z nib souboru *Button_FunViewController.xib*. Aby to tak bylo, nemusíte nic dělat; jde o vlastnost šablony projektu, kterou jste vybrali. V následujících kapitolách vám ukážeme, jak tento proces přesně funguje. Protože soubor *MainWindow.xib* obsahuje ikonu, která reprezentuje `Button_FunViewController`, instance řídícího objektu `Button_FunViewController` se automaticky vytvoří při spuštění vaší aplikace. Když k tomu dojde, tato instance automaticky načte soubor *Button_FunViewController.xib* do paměti a stane se jeho vlastníkem.

Výše jste již do třídy `Button_FunViewController`, která je vlastníkem tohoto nib souboru, přidali outlet. Nyní můžete vytvořit spojení mezi tímto outletem a popiskem prostřednictvím ikony *File's Owner*. Pojďte se podívat, jak se to dělá.

### Poznámka

Pokud zatím proces načítání nib souboru úplně nechápete, je to v pořádku. Tento proces je komplikovaný a budeme o něm mluvit a sledovat ho v několika dalších kapitolách. Prozatím si pouze zapamatujte, že třída řízení je vlastníkem nib souboru se stejným názvem.



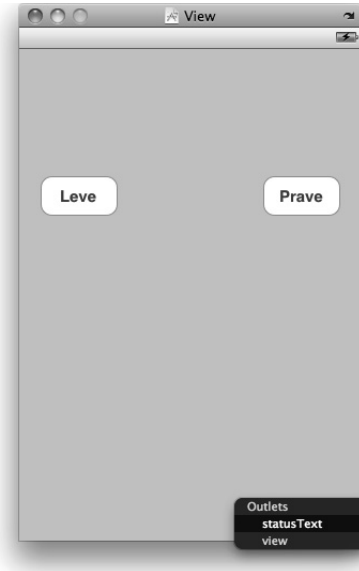

**Obrázek 3.10** Tažení kurzoru se stisknutou klávesou Control

### Propojení outletů

Stiskněte klávesu Control, klepněte na ikonu *File's Owner* v hlavním okně nib souboru a držte tlačítko myši stisknuté. Jeďte s kurzorem z ikony *File's Owner* směrem do hlavního okna *View*. Měla by se objevit modrá pomocná čára. Pokračujte v tažení kurzoru, dokud nebude kurzor nad popiskem v okně *View*. Dokonce i když popisek neuvidíte, najednou se objeví, jakmile nad ním kurzor bude (viz obr. 3.10).

Když je kurzor nad popiskem, uvolněte tlačítko myši a mělo by se objevit malé šedé menu, jako na obrázku 3.11.

Vyberete z menu položku *statusText*.




**Obrázek 3.11** Menu pro výběr outletů

Tažením kurzoru se stisknutou klávesou Control z ikony *File's Owner* nad objekt uživatelského rozhraní jste programu Interface Builder sdělili, že chcete propojit jeden z outletů objektu *File's Owner* s tímto objektem, když se nib soubor načte. V tomto případě je vlastníkem souboru třída `Button_FunViewController` a outlet této třídy, která nás zajímá, je outlet `statusText`. Když jste myší spojili objekt *File's Owner* s objektem popisku a zvolili v menu outlet `statusText`, sdělili jste nástroji Interface Builder, aby propojil outlet `statusText` třídy `Button_FunViewController` s tímto popiskem, takže kdykoliv se budete ve vašem zdrojovém kódu odkazovat na outlet `statusText`, budete pracovat s tímto popiskem. Dobré, že?

### Specifikace akcí

Nyní zbývá pouze identifikovat akce, které tato tlačítka spouští, a za jakých okolností k tomu dochází. Pokud něco víte o programování v Cocoa pro Mac OS X, jste asi připravení stisknout klávesu Control a spojit tlačítka s ikonou *File's Owner* tažením kurzoru opačným směrem. Abychom byli upřímní, můžete to udělat, ale lze to udělat i lépe.

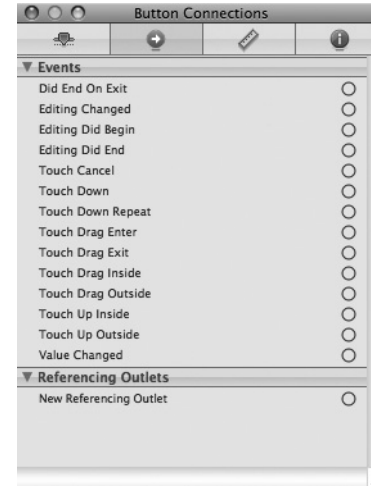
iPhone se od systému Mac OS X liší a toto je jeden z okamžiků, kdy se tato odlišnost projeví. Na počítači Macintosh lze ovládacímu prvku přiřadit pouze jedinou akci, která se typicky provádí ve chvíli, kdy je tento ovládací prvek použit. Z tohoto pravidla sice existují nějaké výjimky, ale obecně ovládací prvek spustí metodu akce ve chvíli, kdy uživatel uvolní tlačítko myši za předpokladu, že kurzor myši se stále nachází nad daným ovládacím prvkem.

Protože ovládací prvky v Cocoa Touch nabízí mnohem více možností, je místo propojení ovládacího prvku s příslušným objektem lepší zvyknout si používat inspektor propojení (connection inspector), který můžete otevřít stiskem kláves ⌘+2, nebo prostřednictvím položky **Connection Inspector** v menu **Tools**. Klepněte na tlačítko *Leve* a otevřete inspektor propojení, viz obrázek 3.12.

Pod hlavičkou *Events* (události) uvidíte úplný seznam událostí, které potencionálně mohou akci spustit. Chcete-li, můžete přiřadit různým událostem různé akce. Můžete například použít událost *Touch Up Inside* (uvolnění tlačítka, když je kurzor stále nad ním) ke spuštění jedné akce a událost *Touch*

*Drag Inside* (stisknutí a potažení uvnitř tlačítka) ke spuštění jiné. Vaše situace je celkem jednoduchá. Když uživatel stiskne tlačítko, chcete zavolat metodu `button:Pressed`. Takže kterou z událostí na obrázku 3.12 máte použít?

Odpověď, která nemusí být na první pohled zřejmá, zní *Touch Up Inside*. Když uživatel odtáhne prst od displeje a pokud se místo, na kterém se displeje v tu chvíli dotýkal, nacházelo uvnitř oblasti tlačítka, provede tuto událost. Vzpomeňte si, co se stane ve většině vašich aplikací pro iPhone, když stisknete tlačítko a na poslední chvíli si to rozmyslíte. Odjedete prstem z tlačítka předtím, než jej uvolníte, že ano? Měli byste vašemu uživateli umožnit to samé. Pokud je však uživatelův prst při uvolnění tlačítka stále na stejném místě, můžete to bezpečně považovat za známku toho, že chtěl tlačítko opravdu stisknout.




**Obrázek 3.12** Použitelné události tlačítka v inspektoru propojení

Když tedy víte, jakou událost použít pro spuštění vaší akce, jak tuto událost propojíte se specifickou metodou akce?

Vidíte v inspektoru ten malý kroužek napravo od položky *Touch Up Inside*? Klepněte do něj, podržte tlačítko myši stisknuté a odjeďte z něj pryč; klávesu Control tentokrát držet nemusíte. Měla by se objevit šedá spojovací čára, podobně jako když jste dříve spojovali outlety. Protáhněte tuto čáru nad ikonu *File's Owner*, a až se objeví malé šedé menu, zvolte v něm položku *buttonPressed:*. Pamatujte, že ikona *File's Owner* reprezentuje třídu, jejíž nib soubor upravujete. V tomto případě ikona *File's Owner* reprezentuje jedinou instanci třídy `Button_FunViewController` vaší aplikace. Když spojíte myší událost tlačítka s ikonou *File's Owner*, říkáte programu Interface Builder, aby se vybraná metoda zavolala ve chvíli, kdy dojde ke specifikované události. Takže když uživatel uvolní tlačítko, aniž by z něj prstem odjel pryč, zavolá se metoda `buttonPressed` třídy `Button_FunViewController`.

Udělejte to stejné s druhým tlačítkem a výsledek uložte. Od této chvíle, kdykoliv uživatel stiskne jedno z tlačítek, zavolá se metoda `buttonPressed:`.

## Testování aplikace

Uložte nib soubor a vraťte se do prostředí Xcode, abyste mohli aplikaci vyzkoušet. Zvolte položku **Build nad Run** v menu **Build**. Váš zdrojový kód by se měl zkompilovat a vaše aplikace spustit v simulátoru telefonu. Když stisknete levé tlačítko, měl by se objevit na displeji text „Leve tlacitko stisknuto.“, jako na obrázku 3.1. Když stisknete pravé tlačítko, měl by se objevit text „Prave tlacitko stisknuto.“.

# Hotovo

Při programování této jednoduché aplikace jste se naučili základy MVC, vytváření a propojování outletů a akcí, implementace objektů řídících vzhled a použití delegátů aplikací. Naučili jste se, jak spouštět metody akcí při stisknutí tlačítka a jak změnit text popisku za běhu programu. Ačkoliv je aplikace, kterou jste naprogramovali, velmi jednoduchá, základní koncepty, které jste při jejím návrhu použili, jsou stejné jako koncepty, na kterých je založené využití všech ovládacích prvků iPhonu,

a ne jenom tlačítek. Ve skutečnosti budete stejným způsobem, jakým jste v této aplikaci použili tlačítka a popisek, implementovat a využívat většinu standardních ovládacích prvků iPhonu.

Je velmi důležité, abyste plně chápali vše, co jsme vám v této kapitole ukázali a proč jste to udělali. Pokud něčemu z výše uvedené problematiky nerozumíte, vraťte se zpět a projděte si příslušnou část znovu. Je to opravdu velmi důležité! Pokud si nejste jistí, že rozumíte všemu, co jste až doposud viděli, budete při navrhování komplexnějších uživatelských rozhraní popsaných v následujících kapitolách této knihy jenom ještě více zmatení.

V následující kapitole se podíváme na některé ze standardních ovládacích prvků iPhonu. Dále se naučíte, jak používat varování a sdělit jejich prostřednictvím uživateli, že se děje něco důležitého, a jak zjistit, že by měl uživatel provést nějakou volbu v menu předtím, než bude pokračovat. Až budete mít pocit, že jste připravení pokračovat, pochvalte se za to, jací jste skvělí studenti, a pokračujte další kapitolou.