Force an iPhone to rotate

Wednesday, 20. March 2013
Letzte Aktualisierung Tuesday, 20. January 2015

How to force an iPhone to rotate to a certain orientation. The Problem iPhone and iPad apps can support portrait or landscape orientation or both. Sometimes a single or a very few of the views needs to be displayed in portrait or landscape only or in that very orientation that the remaining app does not support.   iOS or the cocoa framework respectively provide methods to define individually for each view, which orientations are supported. However, unless any limitations apply to all of the app and are therefore defined on project or target level respectively, it is always the user that has to perform the rotation. The app may decide whether it supports a certain orientation but it cannot force the system to actually rotate.   This tutorial shows how this insufficiency can be overcome.   The example given is for an iPhone app that presents all but one of their view controllers in portrait mode. For good design and usability reasons one of the view controllers works in landscape only.   This is proven on iPhone iOS 6 so far.
Rotating modally presented view controllers This task is easy when the view is presented modally.   For modally presented view controllers, the view controller that is presented should support landscape only and before the presenting view modally, the orientation of the view controller should be set to landscape.   [[UIApplication sharedApplication] setStatusBarOrientation:UIInterfaceOrientationLandscapeLeft animated:YES];[self presentModalViewController:landscapeVC animated:YES]; Doing so, the view controller landscapeVC will be presented in landscape orientation.   But to keep the navigation bar etc. I want to push it to the navigation controller's stack. Unfortunately the device will not rotate its orientation unless the user actually rotates it.   Furthermore, my App has two targets. The navigation of one of them is based on a tab bar and the other target's navigation (the functionality reduces free version) is plain. Therefore my solution needs to suite both, apps with tab bar and apps without.   Therefore this solution may be oversized for an app that has tab bar only or none at all. But it works in any case.   Orientation issue with tab bar controllers   When a tab bar displays its view controllers, then it is the tab bar controller who controls the orientation. Practically that means that all views presented within are supposed to support all the same orientations.   To overcome this we need to subclass UITabBar and introduce some functionality that controls whether the tab bar should support landscape or not.   Changes to the application delegate In my case I added a boolean variable to the application delegate. This is because of the duality of running with tab bar or without. If you have a tab bar only, you might go for adding that variable as property to your tab bar subclass. For OOP off the book a singleton would be nice but I thought that is a bit oversized for a single boolean switch.   AppDelegate.h: @property BOOLÂ Â Â isLandscapePreferred;Â   It is auto-synthesized. Its default is NO. However, you may want to set NO explicitly in application:didFinishLoadingWithOptions: in your app delegate.   For the target that works without a tab bar, all my view controllers need to respond to that setting. Basically the one that is presenting the landscape view must implement the following. But it does not harm having that all over the app especially when there are more than one views in your app that you need to be displayed in landscape.Â   Changes to UIApplication Following some advice that I found on the web, I decided to subclass UIApplication and overwrite supportedInterfaceOrientations there. However, I still marked all orientations as supported in the targetâ€™s Summary pane in xcode, although that setting should be overwritten by UIApplication.   MyApp.h and MyApp.m: #import <UIKit/UIKit.h>@interface MyApp : UIApplication// your interface here @endÂ #import "MyApp.h"#import "AppDelegate.h"@implementation MyApp-(NSUInteger)supportedInterfaceOrientationsForWindow:(UIWindow *)window {Â Â Â if ([(AppDelegate*) [[UIApplication sharedApplication] delegate] isLandscapePreferred]) { Â Â Â Â Â Â Â return (UIInterfaceOrientationMaskLandscapeLeft | UIInterfaceOrientationMaskLandscapeLeft);Â Â Â Â Â Â Â NSLog(@"PhotocollApp: Landscape");Â Â Â } else {Â Â Â Â Â Â Â r
(UIInterfaceOrientationMaskPortrait | UIInterfaceOrientationMaskPortraitUpsideDown);Â Â Â Â Â Â Â NSLog(@"PhotocollApp: Portrait");Â Â }}   @endÂ   Invoking MyApp from main Next we need to make sure that our subclassed applicaton object is instanciated on application launch. That requires a change in the main.m.   #import <UIKit/UIKit.h>#import "AppDelegate.h" #import "PhotocollApp.h"int main(int argc, char *argv[]) {Â Â Â @autoreleasepool {Â Â Â Â Â Â Â return UIApplicationMain(argc, argv, NSStringFromClass([MyApp class]), NSStringFromClass([AppDelegate class]));Â Â Â }}
UIApplicationMain is passed the two received arguments argc and argv in the first two arguments, followed by two NSString objects representing the name of the applicationâ€™s class and an app delegate class. I did not change the standard name of the application delegate but we need to identify our own application class here.   Overwriting supportedInterfaceOrientations in all view controllers I have introduced â€œabstractâ€• subclasses of UIViewController and UITableViewController. All of my view controllers inherit from them:Â   MyRotatingViewController.h and .m: #import <UIKit/UIKit.h>@interface MyRotatingViewController : UIViewController // your interface here
@end
#import "MyRotatingViewController.h"#import "AppDelegate.h"@implementation MyRotatingViewController// Put your code here â€¦
#pragma mark - Rotation ManagementÂ   - (BOOL) shouldAutorotate {Â Â Â return NO;}
- (NSUInteger)supportedInterfaceOrientations {Â Â Â if ([(AppDelegate*) [[UIApplication sharedApplication] delegate] isLandscapePreferred]) { Â Â Â Â Â Â Â return (UIInterfaceOrientationMaskLandscapeLeft | UIInterfaceOrientationMaskLandscapeLeft);Â Â Â } else {Â Â Â Â Â Â Â return (UIInterfaceOrientationMaskPortrait | UIInterfaceOrientationMaskPortraitUpsideDown);Â Â Â }} @end Â   It is exactly the same for MyRotatingTableViewController. The only difference is in the .h file. Naturally it inherits from UITableViewController. #import <UIKit/UIKit.h> Â   @interface MyRotatingTableViewController : UIViewTableController@end Â   Make sure that all (affected) view controllers inherit from MyRotatingViewController or MyRotatingTableViewController respectively. Instead of implementing this â€œabstractâ€• class you may of course implement shouldAutorotate and supportedInterfaceOrientations in all related view controllers.   Â   Custom tab bar class As mentioned above, in tab bar

driven apps, it is the tab bar whose methods control the orientation settings, not those of the view controllers presented by the tab bar.   //Â MyTabBarController.h#import <UIKit/UIKit.h>@interface MyTabBarController : UITabBarController
// your interface here
@end

Â

//Â MyTabBarController.m#import "MyTabBarController.h"   #import "AppDelegate.h"@implementation MyTabBarController- (void)didReceiveMemoryWarning { Â Â [super didReceiveMemoryWarning]; } #pragma mark - Rotation Management- (BOOL) shouldAutorotate {Â Â Â return YES; }

Â

- (NSUInteger)supportedInterfaceOrientations {Â Â Â if ([(AppDelegate*) [[UIApplication sharedApplication] delegate] isLandscapePreferred]) {

Â Â Â Â Â Â Â return (UIInterfaceOrientationMaskLandscapeLeft | UIInterfaceOrientationMaskLandscapeLeft);

Â Â Â } else { Â Â Â Â Â Â return (UIInterfaceOrientationMaskPortrait | UIInterfaceOrientationMaskPortraitUpsideDown); Â Â } }
Â   As I am using storyboard I just need to set MyTabBarController in the properties pane of the one and only tab bar object within IB. If you create it programmatically then just instanciate MyTabBarController instead of UITabBarController.Â   Â   Invoke view controller forced to landscape According to the 80/20 rule we just had 80 percent preparation for the 20 percent real work. J   Now the real work comes â€¦Â   This is in the middle of that very method of a view controller, where the landscape view is pushed. In my case this is in a tableView:didSelectRowAtIndexPath: implementation. It could of course be within an IBAction method or within prepareForSegue:sender: method. If you do that in prepareForSegue then you risk warnings on the error console at runtime. I have not seen any malfunction on one hand but on the other hand I do not know whether this would pass or cause rejection when the app is submitted to the store.Â   The trick is to set the orientation of the status bar to landscape and then to present a view controller modally. Now the device is in landscape mode. The user should not have seen anything so far. Even if, a naked view controller does not have a view. A nil view will not be displayed at all.   Next this recently presented view controller is dismissed. After that the real view controller will be pushed to the navigation stack.   Â   This is the code: //set the landscape switch for the tab bar, view controllers and application

[(AppDelegate*) [[UIApplication sharedApplication] delegate] setIsLandscapePreferred:YES];

Â

//set statusbar to the desired rotation position

[[UIApplication sharedApplication] setStatusBarOrientation:UIInterfaceOrientationLandscapeLeft animated:YES];

Â // Create any view controller. UIViewController will do.

// Present it modally and directly after that dismiss it again.

UIViewController *anyVC = [[UIViewController alloc] init];

[self presentModalViewController: anyVC animated:NO];

[self dismissModalViewControllerAnimated:NO];

// The user should not have noticed anything but how the device is in landscape orientation

Â

// Frankly I am not sure whether the next statement must be included or does not need to be. // While working on â€œmyâ€• solution I came across a number of tricks and hints on several places on the web and this was amongst them.// At least it does not do any harm. if ([UIViewController respondsToSelector:@selector(attemptRotationToDeviceOrientation)]) {Â Â Â // this ensures that the view will be presented in the orientation of the deviceÂ Â Â // This method is only supported on iOS 5.0.Â iOS 4.3 users may get a little dizzy.Â Â Â [UIViewController attemptRotationToDeviceOrientation];}

// Get the storyboard named secondStoryBoard from the main bundle:

UIStoryboard *storyBoard = [UIStoryboard storyboardWithName:@"MainStoryboard_iPhone" bundle:nil];

Â

// I am using storyboard in this project. If you donâ€™t do that then you could just instantiate the view controller the usual way with alloc/init or by loading from nib.

LandscapeVC *landscapeVC = (landscapeVC *)[storyBoard instantiateViewControllerWithIdentifier:@"LandscapeVC"];

Â

// Then push the new view controller in the usual way:[self.navigationController pushViewController:paintingVC animated:YES]; Â   The view controller in landscape orientation We are nearly there. Now the new view controller LandscapeVC is presented in landscape mode on tab bar apps. For apps without a tab bar we have to apply some changes to the LandscapeVCÂ view controller itself. And of course, we need to make sure that the device is rotated back to portrait when the view controller in landscape mode is dismissed.   Â   Sniplets of LandscapeVC.m   #pragma mark - actions// certain tasks need to be performed before the view controller is dismissed. This could be done within the viewWillDisappar.   // Again, if you do that in viewWillDisappear: then you risk some warnings on the console. I have decided to overlay the â€œBackâ€• button (leftBarButtonItem) with a custom button that invokes the following action.   -(IBAction)done :(id)sender{Â Â Â //set the landscape switch back to off/NO/portrait Â Â Â [(AppDelegate*) [[UIApplication sharedApplication] delegate] setIsLandscapePreferred:NO]; Â Â Â //set statusbar to the desired rotation position Â Â [[UIApplication sharedApplication] setStatusBarOrientation:UIInterfaceOrientationPortrait animated:YES]; Â Â Â //present/dismiss viewcontroller in order to activate rotating. Â Â UIViewController *mVC = [[UIViewController alloc] init]; Â Â [self presentModalViewController:mVC animated:NO]; Â Â [self dismissModalViewControllerAnimated:NO];Â Â Â [self.navigationController popViewControllerAnimated:YES];

Â Â Â return;

}
#pragma mark - Rotation// Rotation-(BOOL)hidesBottomBarWhenPushed{ // This one is just for my design. I like to use the full screen of the iPhone for my landscape view controller. Â Â Â Â Â // In the end it is the size of the pane in my case which motivates me to break with UI guidelines and force the user to use this single view controller in landscape only. // Besides this design issue this method is not required for the rotation itself. // What it does: It overwrites the getter of the UIViewController property hidesBottomBarWhenPushed. // By returning YES as constant the bottom bar (= tab bar in my case) will be hidden. // However, it remains hidden in the event that any more view controllers are pushed on top of the navigation stack. // So if you plan to drill further down in your navigation from here, it may not be a good idea to hide the bottom bar. // You will not get it back until the user returns to the calling view controller.Â Â Â return YES;}
// Well, if all of your view controllers inherit from MyRotatingViewController or MyRotatingTableViewController

// respectively then the following is redundant.

// While writing this â€œtutorialâ€• for stack overflow I noticed that this// very view controller does not. Donâ€™t ask me why. I m
it later. // However, I want to show to you what is proven to work fine. Therefore I owe you these methods.

-(BOOL)shouldAutorotate{

Â Â Â return NO;

}

- (NSInteger)supportedInterfaceOrientations{

Â Â Â return (UIInterfaceOrientationLandscapeRight | UIInterfaceOrientationLandscapeLeft);
}

- (UIInterfaceOrientation)preferredInterfaceOrientationForPresentation{

Â Â Â return (UIInterfaceOrientationLandscapeLeft);

} Â   Thatâ€™s it. Pretty straight forward, isnâ€™t it?