# Delegation & Protocols

## Overview

Delegations is a powerful paradigm that is often used to pass information between objects.

Let's say for Example we have an object A, and it wants object B to do some stuff. In order to do that, A will need to have a reference to B, and it will implement the delegate methods of B, that B will call when it is done doing some stuff, or to pass on information to A.

In the example above, A would be considered the delegate, and B would be considers as the delegating object.

Delegates as well as blocks are used heavily by Apple in their API.

Creating your own delegates is simple.

You will need to create a protocol in the header of your delegating object.

```
@protocol nameOfYourDelegate: <NSObject>

//@required is default

  .
  .
  .
  required methods
  .
  .
  .

  @optional
  .
  .
  .
  optional methods
  .
  .
  .

@end

@interface .........
```

Protocols are just a set of rules that define how delegates and the delegating object communicate and pass information between each other. These methods are implemented by the delegates. If the required methods are not implemented and the object confirms to the delegate then the app will crash. Not implementing optional methods won't crash the app. You can think of protocols to be similar to interfaces in languages such as Java and C++.

After defining the protocol you need to create a weak property for the delegate that will link it to the delegating object. `@property (nonatomic, weak) id<nameOfYourDelegate> delegate;`

To make other objects delegates add the  to their interfaces to indicate that they confirm to that classes delegate. Also, don't forget to implement all the required.

There are multiple uses for delegates, you can use them to pass information from: * child to parent. * object to object. * view to object and vice versa. * view to view.

Below is an example of how to create a view to view delegate.

## Implementing the Delegation Pattern

### Project description

Sets the background color in one view by changing the RGB values using sliders in another view

### Set Up

Create a new project in XCode, make it a Single View Application. Create a new view controller and call it SliderViewController.

- Drag in a new View Controller to the Story Board. Create a label and a slider for each RGB value.
- Set the class of the View Controller in the Identity Inspector on the left pane to SliderViewController.
- Link the sliders to SliderViewController using the assistant Editor. Name them redSlider, greenSlider, and Blue slider.
- Add a button below the sliders and call it done.
- Use the assistant editor to create an IBAction for the button named "donePressed".
- Go back to the initial view controller, make sure it is linked to the class XCode created for us. (should be "ViewController").
- Add a button in the middle and label it as "Change Background Color".
- Center it, and make it larger.
- Control drag from the button to the SliderViewController. Set the segue to be Modal and the identifier to "setColor".

Your Storyboard should look like this:

### Creating the protocol

Add the code below to "SliderViewController.h":

```
@protocol SliderViewControllerDelegate <NSObject>
```

```
-(void)changeBackgroundColor: (UIColor *) color;

@end

@interface SliderViewController : UIViewController


//Note the weak property (To prevent cycles)
@property (nonatomic, weak) id<SliderViewControllerDelegate> delegate;

@end
```

The SliderViewController is the delegating object. The @protocol defined the methods that the delegate must to implement. This is because the default enforcement is required, the app will crash if the delegate doesn't implement the methods in the interface. However, if you want some methods to not be required you can just add the @optional tag and add the optional methods below it.

## Conforming to the delegate

Code for "ViewController.m".

```
#import "ViewController.h"
#import "SliderViewController.h"

@interface ViewController ()<SliderViewControllerDelegate>

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}


//The delegate method
-(void)changeBackgroundColor:(UIColor *)color
{
    self.view.backgroundColor = color;
    NSLog(@"Color is %@", color);
};


-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([segue.identifier isEqual:@"setColor"]) {
        //Get the SliderViewController
        SliderViewController *sliderController = [segue destinationViewController];

        //Set up the delegate
        sliderController.delegate = self;

    }
}


@end
```

The code above implements the required delegate methods, and sets it self to be the delegate for the SliderViewController in prepare for segue. Also note that it conforms to the SliderViewControllerDelegate in its @interface.

If you run the project, you'll see that when you press the button and change the sliders and clicked done that the background in the first view controller will have change.

The is a link to the demo that was demonstrated above.

## Links:

- Delegates and Data Sources.
- Understanding Delegation in iOS : An excellent explanation of the details of delegation.
- Stackoverflow: What is Delegation and why is it important.
- iOS Design Patterns.
- Demo Project.