Introduction to Protocol in Objective C

Smitty V Isidhore
Asst.Professor, Department of Computer Science
Carmel College
Mala, Thrissur

Abstract:- **Objective C doesn't have multiple inheritance. All calasses in objective C is inherited from base class Object. Since some problems multiple inheritance is an ideal solution, Objective C provide a tool which make multiple inheritance is not necessary. That tool is called Protocol. In this paper is an attempt to discuss the kind of problem which protocol are designed to solve, how to implement protocol in code and how to design our own protocol. The famous OO design axiom known as ""Favoring composition over inheritance" means that rather than trying inheritance it's always good to solve problem using other classes' functionalities compositely. That is include other class functions in our own classes. A protocol is a group of related properties and methods that can be implemented by any class. They are more dynamic than a normal class interface, since they allow us to reuse a single API declaration in completely separate classes. This makes it possible to denote horizontal relationships on top of a present class hierarchy.**

Key words- Objective c, protocol, objective c protocol, delegate, multiple inheritance in objective c

I. INTRODUCTION

Protocols can be useful in a number of situations, a common practise is to define methods that are to be implemented by other classes. A usual example is when using a tableview, your class implements the cellForRowAtIndexPath method which asks for cell content to insert into a table – the cellForRowAtIndexPath method is defined within the UITableViewDataSource protocol. In delegate design pattern calls for an object that allows a delegate to be assign to it and when object completes its some action its call a method on that object as a call back. Protocol enables us to implement the delegate pattern without forcing a inherence structure. It is one of the most advanced and powerful features of the objective C Language. Protocol allows u to define an interface which other classes will ultimately implement. There is lots of places we can use Protocol, but the most common usage of its in UITableView , UITableViewDataSource and in UITableViewDelegate.[1]

II. PROTOCOL CREATION

Like class interfaces, protocols typically reside in a .h file. To add a protocol to your Xcode project, navigate to File > New> File… or use theCmd+N shortcut. Select Objective-C protocol under theiOS > Cocoa Touch category.

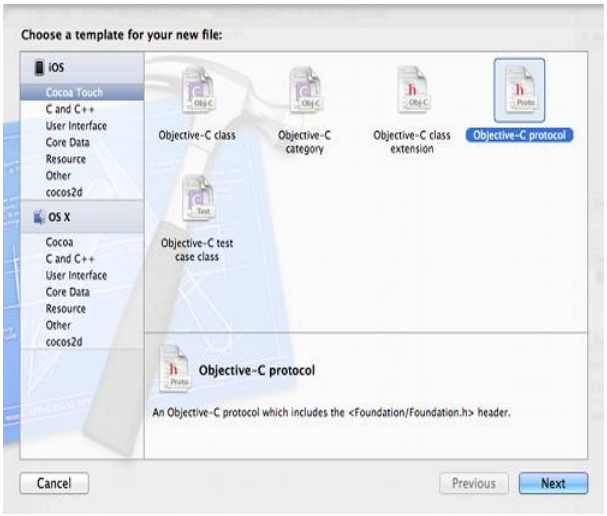

Figure 1: Creating a protocol in Xcode

In this paper, we'll be working with a protocol called RoadRule. Enter this in the next window, and save it in the project root.

Our protocol will capture the necessary behaviors of a street-legal vehicle. Defining these characteristics in a protocol allow us apply them to arbitrary objects instead of forcing them to inherit from the same abstract superclass. A simple version of the RoadRule protocol look like the following**:**

```
// Road Rule.h
#import <Foundation/Foundation.h>

@protocol RoadRule <NSObject>

- (void) signalStop;
- (void) signalLeftTurn;
- (void) signalRightTurn;
@end[4]
```

Any objects that adopt this protocol are assured to implement all of the above methods. The <NSObject> after the protocol name includes the NSObject protocol into the RoadRule protocol. That is, any objects compliant to the RoadRule protocol are essential to obey to the NSObject protocol.[2]

III. ADAPTATION OF PROTOCOL

The above API can be adopted by a class by adding it in angled brackets after the class/superclass name. Create a new classed called Bike and change its header to the following. Note that you need to import the protocol before you can use it.

```
// Bike.h
#import <Foundation/Foundation.h>
#import "RoadRule.h"

@interface Bike : NSObject <RoadRule>

- (void) startRunning;
- (void) removeFrontWheel;
- (void) lockToStructure:(id)theStructure;

@end[4]
```

Implementing the protocol is like adding all of the methods in RoadRule.h to Bike.h. This would work the exact same way even if Bicycle inherited from a different superclass. Multiple protocols can be adopted by separating them with commas (e.g.,<RoadRule, SomeOtherProtocol>).

There's nothing special about the Bike implementation—it just has to make sure all of the methods declared by Bike.h and RoadRule.h are implemented**:**

```
// Bike.m
#import "Bike.h"

@implementation Bike

- (void)startRunning {
   NSLog(@" Here we go!");
}
- (void)removeFrontWheel {
   NSLog(@" Front wheel is off." );
}
- (void)lockToStructure:(id)theStructure {
   NSLog(@ "Locked to structure. ");
}[4]

@end
```

Now, when you use the Bike class, you can assume it responds to the API defined by the protocol.

```
// main.m
#import <Foundation/Foundation.h>
#import "Bike.h"

int main(int argc, const char * argv[]) {
   @autoreleasepool {
      Bike *mybike = [[Bike alloc] init];
      [mybike startRunning];

      [mtbike lockToStructure:nil];
   }
   return 0;
}[4]
```

IV. CHECKING TYPE WITH PROTOCOLS

Just like classes, protocols can be used to type check variables. To make sure an object adopts a protocol, put the protocol name after the data type in the variable declaration, as shown below. The next code snippet also assumes that you have created a Car class that adopts the RoadRule protocol:

```
// main.m
#import <Foundation/Foundation.h>
#import "Bike.h"
#import "Car.h"
#import "RoadRule.h"

int main(int argc, const char * argv[]) {
   @autoreleasepool {
      id <RoadRulel> mysteryVehicle = [[Car alloc] init];
      [mysteryVehicle signalLeftTurn];

      mysteryVehicle = [[Bike alloc] init];
      [mysteryVehicle signalLeftTurn];
   }
   return 0;
}[4]
```

It doesn't matter if Car and Bike inherit from the same superclass—the fact that they both adopt the RadRule protocol lets us store either of them in a variable declared with id <RoadRule>. This is an example of how protocols can capture common functionality between unrelated classes.[3]

Objects can also be verified against a protocol using theconformsToProtocol: method defined by the NSObject protocol. It takes a protocol object as an argument, which can be obtained via the@protocol() directive. This works much like the @selector() directive, but you pass the protocol name instead of a method name, like so:

```
if ([mysteryVehicle
conformsToProtocol:@protocol(RoadRulel)]) {
   [mysteryVehicle signalStop];
   [mysteryVehicle signalLeftTurn];
   [mysteryVehicle signalRightTurn];
}[4]
```

Using protocols in this manner is like saying, "Make sure this object has this particular set of functionality." This is a very powerful tool for dynamic typing, as it lets you use a well-defined API without worrying about what type of object you're dealing with.

V. PROTOCOLS IN THE REAL WORLD

A more accurate use case can be seen in your everyday iOS and OS X application development. The entry point into any app is an "application delegate" object that handles the major events in a program's life cycle. Instead of forcing the

delegate to inherit from any particular superclass, the UIKit Framework just makes you adopt a protocol:

@interface YourAppDelegate : UIResponder <UIApplicationDelegate>

As long as it responds to the methods defined byUIApplicationDelegate, you can use any object as your application delegate. Implementing the delegate design pattern through protocols instead of sub classing gives developers much more leeway when it comes to organizing their applications.

VI. CONCLUSION

In this paper, we conferred another structural tool in Objective C. Protocols are a technique to abstract shared properties and methods into a dedicated file. This benefits minimize the repeated code and allow us to check dynamically if an object supports an arbitrary set of functionality. Cocoa frameworks have many protocols. A common use case is to help us to alter the behaviour of certain classes without the need to subclass them. For instance, the Table View, Outline View, and Collection View UI components all use a data source and delegate object to configure their internal behaviour. The data source and delegate are defined as protocols, so you can implement the necessary methods in any object you want.

REFERENCES

[1] Kochan Stephen G, Programming In Objective-C: A Complete Introduction To The Objective-C Language 1 Jan 2004
[2] Stephen Kochan ,Programming in Objective - C, 4e Paperback – 2012
[3] Matt Neuburg, iOS 7 Programming Fundamentals Paperback – 1 Nov 2013
[4] RyPress, Ry's Objective –C Tutorial , "Protocols" [nd]. Avilable: http:// rypress.com/ tutorials/objective-c/protocols