Abstract:

This Application Note demonstrates the setup and implementation of new classes within the Xcode environment providing insight into Objective-C coding and the Xcode Integrated Development Environment.
Outline

Introduction

Project Setup and Classes Tutorial

Tips and Recommendations

References
**Introduction**

Many iOS applications require new classes to be written and implemented to do the actual work of the app. This Application Note focuses specifically on creating new files that demonstrate the structure of integrated classes throughout the program. What this Application Note does not go over is the final view of the application, but the work of the classes gets done internally. the implementation of various header and implementation files.

**Project Setup and Classes Tutorial**

In a new Xcode single-view project, header (.h) and implementation (.m) files will be automatically created as they provide the structural support for the application. Many of these files can (and should) be added to in order to make a well-rounded application that has purpose in a world with multiple apps being released every day. The names of the present files at the beginning of a new project are Main.Storyboard, AppDelegate.h, AppDelegate.m, ViewController.h and ViewController.m. Main.Storyboard is an abstract file that is viewed just as a user will view the app later much like a cartoon storyboard. Appropriately, it is referred to as a View file. The files with the same name except for the extension work together and create what we call in computer science a “class.” Different types of objects are defined within different “classes” which serve as blueprints for any number of objects within reason. Let’s build classes to do some work in an
To briefly outline what the default files do, the AppDelegate class deals with how the app relates to the iPhone when the application is first opened, terminated or running either in the foreground or background. The ViewController class controls the different pages or views that the app displays. The next class we are going to make ourselves and it is called a Model class. Model classes describe what happens with newly created objects within your app.

We are going to create a simple Model class called Adder that will internally add two integers together, but we are not going to display it.
After Xcode has been opened to a new project which should look like this...

Figure 2. New Xcode Project

Add new files by navigating to File/New/File...

Choose Objective-C Class and Click Next.

Figure 3. A New Class
Next, we get to name our new class. We will name it “Adder” and make sure that it is a subclass of NSObject which just means our new class is inherited from NSObject so it has some of the same methods. Click Next when done.

![Choose options for your new file:](image)

**Figure 4. New Class Name**

After this step, you will notice that both a new .h and .m file have been created and classes usually work in pairs like this.
Figure 5. New Files

Now we can add content to these files to have them do some work – in our case, add numbers together. The following is all the code that needs to be put into the Adder.h file:

```c
#import <Foundation/Foundation.h>
@interface Adder : NSObject
{
    int number1;
    int number2;
}
-(void) setNumber1: (int) n1;
-(void) setNumber2: (int) n2;
-(int) addition;
@end
```
It is apparent that the above code provides the definitions for the different parts of our Adder object. The “#import” line lets this file have access to what is defined in files not yet included into our project. The “@interface Adder : NSObject” line describes attributes of a class – in our case, two numbers that will be taken as inputs.

In the file Adder.m copy the following code that acts as the implementation of the declared files from Adder.h and notice that we “#import” Adder.h first in order to have access to the declarations.

```
#import "Adder.h"
@implementation Adder

-(void) setNumber1:(int) n1
{
    number1 = n1;
}

-(void) setNumber2:(int) n2
{
    number2 = n2;
}

-(int) addition
{
    return number1+number2;
}
@end
```
The “@implementation” line does precisely what it seems like it should. It begins the section of giving the Adder class the work it has to do. The rest of the lines are relatively self-explanatory if you are familiar with coding, but especially notice where the addition actually takes place near the end.

```c
-(int) addition
{
    return number1+number2;
}
```

This app is not quite ready be built for several reasons, but first let’s add to a file we have not yet talked about. Another automatically created file is the main.m file which is where the actual app runs from and it calls the classes from this central or main place. It’s found under Supporting Files.
Adding this code to main.m text space will be the last bit we can accomplish short of building an actual view.

```c
#import <UIKit/UIKit.h>
#import "AppDelegate.h"
#import "Adder.h"

int main(int argc, char *argv[]) {
    @autoreleasepool {

        Adder *calculator;
        calculator = [Adder alloc];
        calculator = [calculator init];
        [calculator setNumber1: 35];
        [calculator setNumber2: 65];
        NSLog(@"Hello, the sum is %i", [calculator addition]);
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
    }
}
```

Note that Adder.h is being imported at the top.

Now press the Play symbol in the upper left-hand corner of the window and after a while it should read “Build Succeeded.”

That completes the tutorial on adding new types files to Xcode. The next step is linking up the ViewController to display the content and then move towards an interactive interface where users can input their own numbers.

This relates to our group project because my specific part focuses on implementing the new classes that include the class data that translates the speech-to-text and text-to-speech.
**Tips and Recommendations**

It is recommended to look at many coding examples from Objective-C before beginning to write custom classes because the syntax is not always intuitive. If there are errors along the way in Xcode, the IDE uses red marks or yellow warnings on the left of the text field to indicate that there is an issue that will prevent it from compiling. There are plenty of resources on the world wide web for gaining more knowledge about any particular piece of creating classes, because depending on how complex the methods are, there will be more potential roadblocks. See **References** for excellent website resources.
References

Model Object:


Project Files Explained:

http://codewithchris.com/xcode-project-files-explained/

Any type of coding question forum:

http://stackoverflow.com/