React Native iOS Communication between native and React Native

Introduction

React Native is a new framework that helps mobile developers build mobile apps with Javascript and native languages. It uses the same design as React, letting developers compose a rich mobile UI from declarative components.
This document is about how to implement the communication between react native and native.

Problems And Approaches

1. Call native functions from react native

Problem:

Sometimes an app needs access to platform API, and React Native doesn't have a corresponding module yet. Maybe you want to reuse some existing Objective-C, Swift or C++ code without having to re-implement it in JavaScript, or write some high performance, multi-threaded code such as for image processing, a database, or any number of advanced extensions.

Solution:

React Native is designed to supports developers to write real native code and have access to the full power of the platform. If React Native doesn't support a native feature that developers need, then developers should be able to build it themselves.
Implement:
Implement a native module which conforms RCTBridgeModule

```
// CalendarManager.h
#import "RCTBridgeModule.h"
@interface CalendarManager : NSObject <RCTBridgeModule>
@end
```

Export this module using RCT_EXPORT_MODULE function

```
 // CalendarManager.m
@implementation CalendarManager

RCT_EXPORT_MODULE();

@end
```

Export module’s functions

```
#import "CalendarManager.h"
#import "RCTLog.h"

@implementation CalendarManager

RCT_EXPORT_MODULE();

RCT_EXPORT_METHOD(addEvent:(NSString *)name location:(NSString *)location)
```

Call native module’s function in react native

```
import { NativeModules } from 'react-native';
var CalendarManager = NativeModules.CalendarManager;
CalendarManager.addEvent('Birthday Party', '4 Privet Drive, Surrey');
```

Conclusion:

This solution is supported by React Native and it’easy to implement. Developers can implement many native modules which wraps their native functions or platform functions then import and call them in React Native.

2. Call or send events to React Native from native

Problem

When developing apps using react native, we want to develop our native components which have better performance or support native functions. We also want to send data, information from many places in native scope to react native and handle these data, information. This section defines two approaches for these problem.

I. Implement Native Component:

Beside supporting native module implementation, react native also supports developers to define their own native components. React Native document has a good doc for implement instruction. In this section we will focus on how to send data from native modules to react native.

- **Implement:**

To send data from a native module to react native, the native module must define a property which has type RCTBubblingEventBlock

```
@protocol EventViewDelegate;

@interface EventView : UIView

@property (weak, nonatomic) id<EventViewDelegate> delegate;

@property (nonatomic, copy) RCTBubblingEventBlock
onEventHandler;
```

In the View Manager class of native component, we need to export this property and call block function when needed

```
@interface EventViewManager()<EventViewDelegate>

@end

@implementation EventViewManager

RCT_EXPORT_MODULE()

- (UIView *)view {
  EventView *eventView = [EventView new];
  eventView.delegate = self;
 return eventView;
}

RCT_EXPORT_VIEW_PROPERTY(onEventHandler,
RCTBubblingEventBlock)


- (void)eventTrigger:(EventView *)eventView withParams:
(NSDictionary *)params {
  dispatch_async(dispatch_get_main_queue(), ^{
    if (eventView.onEventHandler) {
      eventView.onEventHandler(params);
    }
  });
```

And finally, in react native we define a wrapper component and handle the block function

```
class EventComponent extends Component {

    constructor() {
        super();
        console.log("event component was created");
        this.eventHandler = this.eventHandler.bind(this);
    }

    eventHandler(event) {
        if (this.props.handleEvent) {
            this.props.handleEvent(event.nativeEvent);
        }
        console.log("event block call with params", arguments);
        alert("event block call with params", arguments);
    }

    render() {
        const eventView = (
            <EventView
                onEventHandler={this.eventHandler}
                style={{
                    width: 100,
                    height: 100,
                    backgroundColor: "blue"
                }}/>
        );
        return eventView;

    }
```

- **Conclusion:**

This approach is supported by React Native. It's rather complicated to implement since it requires many steps.
The handler function/event function property is sent only from native module which own it to react native wrapper component so It cannot be called out side the scope of the owner component.

II. Implement an event emitter

The main purpose of this approach is sending event and data from any places in native to react native. This approach is not documented by React Native but by taking the advantage of RCTEventEmitter class, we can define classes which help us to achieve our goal.

- **Implement:**

We define a native module which inherits from RCTEventEmitter

```
@interface EventEmitter : RCTEventEmitter

+ (void)dispatchAnEvent:(NSDictionary *)eventInfos;

@end
```

We must override these functions of RCTEventEmitter

```
- (NSArray<NSString *> *)supportedEvents {
  return @[@"sayHello"];
}

- (void)startObserving {
  for (NSString *event in [self supportedEvents]) {
    [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(handleNotification:)
                                                   name:event
                                                 object:nil];
  }
}

- (void)stopObserving {
  [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

Then we implement our sending event functions

```
+ (void)dispatchAnEvent:(NSDictionary *)eventInfos {
  [[NSNotificationCenter defaultCenter]
postNotificationName:@"sayHello" object:NULL
userInfo:eventInfos];
}

- (void)handleNotification:(NSNotification *)notification {
  [self sendEventWithName:@"sayHello"
body:notification.userInfo];
```

Add listener for React Native code where we need to handle the event

```
const myModuleEvt = new NativeEventEmitter(NativeModules.EventEmitter)
myModuleEvt.addListener('sayHello', (data) => console.log(data))
```

Finally in native code, we call the sending event function

```
- (void)touchesEnded:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
  [EventEmitter dispatchAnEvent:@{@"Touch Ended": @"Hit"}];
}
```

NOTES:
THIS IS HOW THIS CLASS WORKS:

+ The event emitter defines what events it supports.
+ When this class is created it starts observing the events which are defined above. This is the trick, when we import this module in the react native this module will be created automatically, we don’t have to create this module and send to react native as bridge module

- **Conclusion:**

By using this approach we can send event with data to react native from any places in native. Implement event emitter is easy. But the limit of this approach is every object in react native which subscribes the event of emitter will invoke the handler function each time that event is sent.
Event Emitter implementation has the same concept with NSNotificationCenter of iOS platform.

Summarize:

Sending and handling event and data from/to React Native to/from native are most needed features which are used regularly in app development. Depend on the features and functionalities of the app we should choose the proper solution - one of two above - to handle the communication between native and react native