iOS Framework API Guide

The iOS Framework provides a simple mechanism to connect to an OpenXC VI via an iOS app. The iOS Framework is written in Swift2.

Initializing the VehicleManager

```
var vm: VehicleManager!
vm = VehicleManager.sharedInstance
```

Configuring the VehicleManager

```
// TODO – allow query of detected VIs (optional)

// TODO – allow setting to protobuf vs json (optional, json is default)

// set callback to receive manager status update (eg. VI detected, BLE connected,
// disconnected, message parse error, etc)
// (optional)
  vm.setManagerCallbackTarget(self,
    action: ViewController.manager_status_updates)
 elsewhere
  func manager_status_updates(rsp:NSDictionary) {
    let status = rsp.objectForKey("status") as! String
    print("VM status : ",status)
  }

// set to display log output
// (optional)
  vm.setManagerDebug(true)
```

Tell VehicleManager to connect to VI after config options

```
  vm.connect()   →   auto connects to first discovered VI
    or
  vm.connect(peripheral) → if connection to specific VI is required
```

MEASUREMENT MESSAGES

Reading the last recorded value for a specific measurement

```
let rsp = vm.getLatest("fuel_level")
print("FUEL_LEVEL value:",rsp.value)
print("FUEL_LEVEL timestamp:",rsp.timestamp)
```

Add a callback function for any measurement response of a specific type

```
vm.addMeasurementTarget("ignition_status",
        target: self,
        action: ViewController.display_ignition_status_change)
```

elsewhere:

```
func display_ignition_status_change(rsp:NSDictionary) {
  let vr = rsp.objectForKey("vehiclemessage") as!
    VehicleMeasurementResponse
  print("specified meas:",vr.name," -- ",vr.value!)
}
```

Add a default callback function for any measurement response that hasn't been specifically added as above

```
vm.setMeasurementDefaultTarget(self,
  action: ViewController.default_measurement_change)
```

elsewhere: (note handling of evented vs non evented messages)

```
func default_measurement_change(rsp:NSDictionary) {
  let vr = rsp.objectForKey("vehiclemessage") as!
         VehicleMeasurementResponse
  if vr.isEvented {
    print("default meas(evented):",vr.name," -- "
               ,vr.event," -- ",vr.value)
  } else {
    print("default meas:",vr.name," -- ",vr.value)
  }
}
```

Remove a callback for a specific measurement

```
vm.clearMeasurementTarget("fuel_level")
```

Remove the default callback for measurement responses

```
vm.clearMeasurementDefaultTarget()
```

COMMAND MESSAGES

Send a Command message with a response callback

```
      let cmdcode = vm.sendCommand(.version,
            target: self,
            action: ViewController.display_version_rsp)
      print("version cmd sent:",cmdcode)
```

elsewhere:

```
func display_version_rsp(rsp:NSDictionary) {
  let vr = rsp.objectForKey("vehiclemessage") as!
    VehicleCommandResponse
  let code = rsp.objectForKey("key") as! String
  print("cmd_rsp \(code) : \(vr.show())")
}
```

cmdcode returned from vm.sendCommand can be used as an identifier to match up command responses if a common command response handler is used. Command responses also return in the same order that the commands themselves are sent in.

Send a Command message with no callback

```
      vm.sendCommand(.device_id)
```

TRACE FILES

Setting up a Trace file sink

```
vm.enableTraceFileSink("tracefile.txt")
```

- generally called before vm.connect
- calling this method will erase the trace file supplied if it already exists
- the client ios app must include `UIFileSharingEnabled=true` *in Info.plist*
- the resulting trace file is accessible via iTunes File Sharing

→ *with ios device plugged in to computer, open itunes. Select device, open Apps tab, scroll to bottom. Select openXC app. Configured trace file can be downloaded to computer.*

```
vm.disableTraceFileSink ()
```

- turns off file sink

DIAGNOSTIC MESSAGES

W.I.P.