Redpark

Serial Cable SDK
Users Guide

1 Introduction

This document explains the library and the example interface code that are supplied as part of the SDK. Both this document and the example interface code are intended for developers who are experienced in writing iOS applications and who have a reasonable understanding of serial data communications.

While Redpark can offer further explanation of the example interface code that has been provided, we cannot offer assistance in debugging a specific app or provide general advice regarding the topic of iOS app programming.

1.1 Overview

The SDK contains an objective-C class “RscMgr” (Redpark Serial Cable Manager) which encapsulates and abstracts away the details of the Redpark Serial Cable protocol to simplify and ease application development.

The RscMgr class provides a basic init,setBaud,open,read,write... style of interface to the serial port. To use the RscMgr, you should include “RscMgr.h” and “libRscMgr.a” into your project.

It’s assumed you are already familiar with writing cocoa applications for iOS devices and how objective-c delegate protocol interfaces work.

Also included in this release is a sample iOS application “RSC Demo” which demonstrates how to use the RscMgr class to communicate with the RSC serial port. Look at “RootViewController.h” and “RootViewController.m” in the “RSC Demo” project folder for an example of how to create and use the RscMgr class.

2 Notes for Developers

2.1 Using the Redpark Serial Cable Manager

You should only create one instance of the RscMgr inside of the application and call init. The RscMgr will then register for the appropriate notifications from the iOS to detect when an accessory is connected or disconnected. As of iOS 4.2, disconnect and connect notifications are also sent when your application is moved to the background and foreground. These should be dealt with the same way as a physical disconnect or connect of the RSC serial port. If the attached accessory matches one of the protocols supplied to initWithProtocol then RscMgr will call the cableConnected callback.

The RscMgr class requires another objective-C object to be designated as its delegate to respond to various events related to the cable (i.e. cableConnected, cableDisconnected, readBytesAvailable...). It is customary for the class which instantiates the RscMgr to designate itself as the delegate. Please look at viewDidLoad in RootViewController.m for an example of the initialization process.

Once connected, the application can open a connection to the serial port and begin data communication. At any time, the application may change the port configuration by calling the various “set” routines such as setBaud, setDataSize, setParity, and setStopBits.

The application must call open before calling write or read. The delegate object will receive the readBytesAvailable callback when serial data has arrived. The delegate class should call read inside of this callback to get the available data and to avoid an overrun. Look at the readBytesAvailable: function in RootViewController.m. The RscMgr class will buffer up to 1024 bytes before an overrun occurs (see RscMgr.h). The user can modify the value kRSC_SerialReadBufferSize to change the buffer size.

The delegate class will also receive port status updates through the portStatusChanged callback. The application may wish to query modem signal states at this time using the getModemStatus accessor. In the case where flow control is used, the application should call getPortStatus and query the specific bits in the serialPortStatus structure (see redparkSerial.h).

In addition to the basic serial port configuration options (setBaud, setDataSize, etc...) a developer may also access some additional features using the serialPortConfig and serialPortControl structures directly (see redparkSerial.h). However, developers should be cautious enabling these "advanced" features unless they know this is required for their device.

Please note, the txFlush and rxFlush options will purge the TX/RX internal serial buffers. This is not the same as flushing an iostream which results in forwarding buffered bytes. In general, the Rsc accessory will immediately forward TX bytes received from the iOS device unless there is a halt condition (i.e. related to flow control).

Byte forwarding in the receive direction is controlled by the rxForwardCount and rxForwardingTimeout options. The Rsc accessory will forward received bytes to the iOS device when either rxForwardCount bytes have arrived or the line is idle and rxForwardingTimeout expires. These rx forwarding options can be customized to suit the protocol needs of the serial device you are communicating with. Increasing the rxForwardCount will buffer more data in the cable and utilize larger packets over the link between the Rsc accessory and the iOS device. For protocols with large message sizes and where high throughput is desired increasing the rxForwardCount may increase performance. However, protocols that exchange small messages (a few bytes at a time) may want to decrease the rxForwardCount and or the rxForwardingTimeout. Some experimental "tuning" may be required. Warning, for high throughput communication, reducing the rxForwardCount could degrade performance significantly because of increased packet overhead related to the accessory to iOS link.

You should keep in mind that the RscMgr in and out streams are scheduled on the application's RunLoop. Therefore, any writes or reads are asynchronous and may not be processed until your application has returned to the RunLoop. This is only an issue for applications that require precise transmission and receive timing. The application should use the readBytesAvailable delegate function to determine if data has been received.

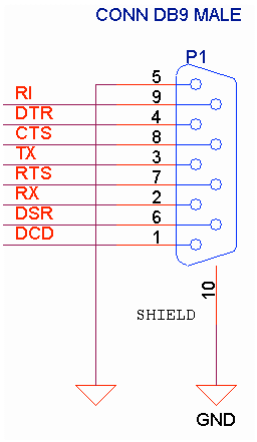
2.3 Rsc Demo

The Rsc Demo application project displays a simple UI allowing the user to change port configuration settings and run a loop back test (Redpark loopback adapter required). It is assumed you are familiar with standard cocoa UITableView's. This is a common UI element in iOS Device applications for displaying configurable settings to a user. The RootViewController class sub-classes from UITableViewController and is a delegate of the RscMgr class. It creates the table cells based on information retrieved from the RscMgr's port configuration. Look at getPortConfigSettingText and setPortConfigSettingFromText as examples of how to get and set the port configuration. The application displays the status of each modem signal and allows the user to raise/lower the RTS and DTR modem signals.

3 Apple Documentation

Please refer to the “External Accessory Programming Guide” available in the iOS Resource Library on the iOS developer website for additional guidance.

4 DB-9 Connector Pin Out



Pin Number	Function	I/O	Description
1	DCD	I	Data carrier detect
2	RX	I	Receive
3	TX	O	Transmit
4	DTR	O	Data terminal ready
5	GND	PWR	Ground
6	DSR	I	Data set ready.
7	RTS	O	Request to send
8	CTS	I	Clear to send
9	RI	I	Ring indicator