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 (R261). 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 the SDK 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.

1.2 “What’s New?”

This release of the SDK requires iOS 5.1 (or later) and includes a new sample app – “Rsc Thread Demo”. This example demonstrates how to put the RscMgr communication handling on a different thread (other than the Main UI thread). The RscMgr will schedule its stream handling on the run loop of the thread where init is called. Likewise, delegate callbacks will happen on that thread as well. This allows the application to perform synchronous processing of received data without blocking the UI thread.

This release of the SDK includes several new convenience access functions that allow the caller to read/write data as NSString or NSData:

```
// same as write: but takes an NSData object instead of a C style buffer
- (void) writeData:(NSData *)data;

// Send a string over serial connection - same as write: but takes an NSString object
// Note - uses UTF8String to convert/retrieve bytes before sending and includes \0
character
- (void) writeString:(NSString *)string;


// returns an NSString containing the available bytes in rx fifo as a string.
// assumes the data is ASCII and encoded as UTF8
// calling this clears rx fifo similar to read:
- (NSString *) getStringFromBytesAvailable;

// returns an NSData containing the available bytes in rx fifo
// calling this clears rx fifo similar to read:
- (NSData *) getDataFromBytesAvailable;
```

This release of the SDK also adds a new function `supportsExtendedBaudRates` which allows you to determine if the connected cable supports baud rates > 57600. See section 2.4 “Supporting the new C2-DB9V Serial Cable” for more details.

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.

A new key needs to be added to the application plist “Supported external accessory protocols”, with Item 0’s value being “com.redpark.hobdb9” and Item 1’s value as “com.redpark.hobdb9v”. Look at the RSC Demo application’s plist for an example.

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). This is independent of the receive fifo in the cable itself. See below for a discussion of the cable’s receive forwarding logic.

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).

Advanced Options:

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.

To facilitate **large continuous data transfers,** you should turn on the **txAck** option in the port configuration structure. The cable will send a transmit acknowledgement bit in the port status structure when the cable's internal TX FIFO is empty. The TX FIFO size is defined in RscMgr.h (currently 256 bytes) so you can make sure not to write more than that amount at a time. After sending the first chunk, your application should wait to receive a txAck before sending the next blocks. This will prevent overrunning the cable's TX Buffer. See the **Rsc Demo** application for an example of how to do this. In its loopback test it uses the txAck feature to send a large block of data (i.e. > 256) where `LOOPBACK_TEST_LEN = 4096`.

Run Loop:

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.

2.4 Supporting the new C2-DB9V Serial Cable

We recently released a new C2-DB9V serial cable that supports baud rates up to 115.2 Kbps. If you have already written an iOS app that works with our C2-DB9 serial cable, you will need to take two steps to revise the app so that it is compatible with the new C2-DB9V serial cable.

1. Rebuild your project using the library contained in this release of the SDK.
2. Add the "com.redpark.hobdb9v" ea protocol name to the plist in your application.
3. Optional - Call `supportsExtendedBaudRates` to figure out if the connected cable supports 115.2 Kbps. This would only be necessary if you wanted to support the old and the new cable and use the higher baud rate when available.

3 Useful Resources

3.1 Apple Reference Materials

The “External Accessory Programming Guide” available in the iOS Resource Library on the iOS developer website provides useful guidance.

3.2 Sample Projects

In addition to the RSC Demo and RSC Thread Demo projects included in our SDK, you will find that some of your fellow developers have chosen to share their projects on GitHub (www.github.com). Search for “Redpark” on the GitHub site. (Please note that these projects have not been reviewed by Redpark staff.)

3.3 Make Magazine

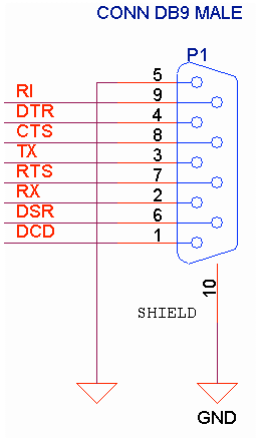
The editors at Make Magazine (www.makezine.com) have posted step-by-step projects that are particularly useful for beginning developers. Search for “Redpark” on the Make Magazine site.

3.4 FAQ

Redpark does not provide assistance in debugging your code. However we maintain a list of frequently asked questions that includes programming topics. Please see..

http://www.redpark.com/c2db9_FAQs.html

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