Sample Code

Communicating with a Modem on a Serial Port

Find and connect to a modem attached to a serial port using IOKit.

Download

macOS 10.6+ | Xcode 11.2+

Overview

This sample project creates a command-line program that uses IOKit to discover a modem on a serial port and to send and receive data. Run the program within Xcode and use the console to monitor the progress of the communication.

Discover Attached Modems

IOKit provides a matching service for various types of I/O devices, including serial ports and modems. The sample code's `findModems()` function uses `kIOSerialBSDServiceValue` to locate a serial device. Once located, the code identifies the type of serial device as a modem with `kIOSerialBSDModemType`.

```
classesToMatch = IOServiceMatching(kIOSerialBSDServiceValue);
if (classesToMatch == NULL) {
    printf("IOServiceMatching returned a NULL dictionary.\n");
}
else {
    // Look for devices that claim to be modems.
    CFDictionarySetValue(classesToMatch,
                         CFSTR(kIOSerialBSDTypeKey),
```

```
                        CFSTR(kIOSerialBSDModemType));
}

// Get an iterator across all matching devices.
kernResult = IOServiceGetMatchingServices(kIOMasterPortDefault, classesToMatch, mat
```

The `findModems()` function returns an IOKit iterator that `getModemPath()` uses to locate the first available modem and obtain its formal BSD device path.

If the function fails to find a modem or can't obtain the device, the program prematurely quits.

Open the Serial Port and Initialize the Modem

After the `findModems` function finds the modem, `openSerialPort()` uses the low-level `open()` call to connect to the BSD file path discovered earlier. This function also captures the state of the serial port and saves it for later restoration.

```
fileDescriptor = open(bsdPath, O_RDWR | O_NOCTTY | O_NONBLOCK);
if (fileDescriptor == -1) {
    printf("Error opening serial port %s - %s(%d).\n",
           bsdPath, strerror(errno), errno);
    goto error;
}
```

This code allows the function to set up communication parameters such as baud rate, blocking characteristics, and handshaking lines.

If the call succeeds, it returns a BSD file descriptor. The `initalizeModem()` function then takes the file descriptor and sends the attention (AT) command to the modem using the low-level `write()` function. It then calls the `read()` function to obtain a response that confirms the modem is present.

Close the Serial Port

The `closeSerialPort()` function takes the passed file descriptor and waits for all output to flush, then restores the serial port to its original state and closes the file descriptor.

```
if (tcdrain(fileDescriptor) == -1) {
    printf("Error waiting for drain - %s(%d).\n",
           strerror(errno), errno);
}
```

```
// Traditionally it is good practice to reset a serial port back to
// the state in which you found it. This is why the original termios struct
// was saved.
if (tcsetattr(fileDescriptor, TCSANOW, &gOriginalTTYAttrs) == -1) {
    printf("Error resetting tty attributes - %s(%d).\n",
           strerror(errno), errno);
}

close(fileDescriptor);
```

This action terminates communication to the modem and concludes the use of the serial port.