



UPhil Programming Guide

Document Version 2.0

Prepared by:



WebSec Corporation
41531 Date Street
Murrieta, CA 92562

Table of Contents

1	UPHIL FEATURES.....	1
2	(U) UPHIL CONFIGURATION.....	2
3	PC SYSTEM REQUIREMENTS.....	3
4	WINDOWS PROGRAMMING WITH UPHIL.....	4
4.1	CUSTOM COMMANDS WITH UPHIL	4
4.1.1	Setting baud rate	4
4.1.1.1	Examples for Setting Baud Rate.....	4
4.1.2	Set mode	5
4.1.3	Command For Asynchronous COMS (RS232)	5
4.1.3.1	Stop bits	5
4.1.3.2	Parity	5
4.1.3.3	Data bits	6
4.1.4	Setting LEDs.....	6
4.1.4.1	Red LED.....	6
4.1.4.2	Green LED	6
4.1.4.3	Blue LED	6
4.1.5	Other commands	7
4.1.5.1	Flip Polarity on Data bits	7
4.1.5.2	Sending DS-101 Wakeup.....	7
4.1.5.3	Receiving DS-101 Wakeup	7
4.1.5.4	Send DS-102 CTS.....	7
4.1.5.5	Receive DS-102 CTS.....	8
4.1.5.6	Auto Control DS-102 CTS.....	8
4.1.5.7	DS-102 CFD Multiplex data	8
4.1.5.8	DS-102 Bit Inversion (RX/TX)	8
4.1.5.9	Read FPGA Version	8
4.1.5.10	Connect Fill Port and USB shield	8
4.1.5.11	Active Zeroize Buffers.....	9
4.1.6	Example Command Sets.....	9
4.1.6.1	Example to Set Up Communication Using DS-101	9
4.1.6.2	Example to Set Up Communication Using DS-101 Wakeup-On	9
4.1.6.3	Example to Set Up Communication Using DS-232	9
4.1.6.4	Example to Set Up Communication Using DS-102	9
4.2	APPLICATION DEVELOPMENT	9
4.2.1	RS-232	9
4.2.1.1	RS-232 Sample Code.....	10
4.2.2	DS-101	10
4.2.2.1	DS-101 Sample code.....	11
4.2.3	DS-102	11
4.2.3.1	Sample Code to Send a Test Key via DS-102 with CTS to an ECU	11
4.2.3.2	Sample Code to Receive a Key via DS-102, including a CTS command	13
4.2.3.3	Sample Code to Load and A and B Segment.....	18

5	LINUX PROGRAMMING WITH UPHIL	25
6	ADDITIONAL INFORMATION.....	26
6.1	FAQs	26
6.2	SUPPORT	26
6.3	DATA COLLECTION DEVICE (DCD)	26
6.4	EKMS SOFTWARE LIBRARIES	26

1 UPhil Features

UPhil is a USB to DS-101 / DS-102 / RS-232 adapter designed specifically for the EKMS and KMI community.

UPhil features include:

- DS-101, DS-101 with wakeup, and DS-102 modes, including Clear-to-Send (CTS)
- Can be used as a primary (sender) or secondary (receiver) station
- Optical switching for maximum signal isolation and security
- Active and passive zeroization
- Can be utilized for TEMPEST or TRANSEC testing
- Allows control and data to be sent using standard byte level serial communications (e.g., COM4)
- Ability to change protocol modes on the fly with simple programming commands or included application
- Supports RS-232 speeds up to 115,200 baud and above; and DS-101 speeds over 64K baud
- Simple, well-defined API
- Optimized for use with existing programs such as ACES, DMD, iApp, and the SKL Update Wizard to significantly reduce database, audit, or software transfer times
- SKL UAS upload transfer times via fill port are reduced from 5 hours to 50 minutes
- Passively zeroizes firmware and memory when disconnected
- Allows for UAS uploads to the SKL and other EKMS fill devices in SCIFs where USBs are not authorized
- Adapter supports all current Tier 3 Fill Devices and is compatible with next generation Fill Devices, such as the Air Force Portable Key Loader (PKL)
- OS support includes Windows, Linux, and Android

2 (U) UPhil Configuration

The standard UPhil configuration includes the following:

1. UPhil Dongle
2. Standard USB Type A to mini B cable
3. UPhil Software (CD-Rom)



3 PC System Requirements

UPhil requires the following minimum PC features:

- Windows® 7,8, or 10 32-bit or 64-bit operating system
- Pentium or higher processor
- One USB 3.0 Super Speed or USB2.0 High Speed enabled port. It will not run on USB 1.1 Full Speed ports
- 50MBytes of Hard disk space
- 32MBytes of RAM

4 Windows Programming with UPhil

UPhil functionality can be extended by incorporating the following custom programming options. These options are recommended for experienced developers and can be written in numerous languages, including Perl, C, C++, C#, Python, etc. For purposes of this guide, our examples will be written in C++.

With UPhil attached to both the PC and to an end unit (e.g., SKL, TKL), the user can program UPhil to perform advanced EKMS protocols. The types of EKMS protocols that the user can customize are all modes of DS-101; all modes of DS-102 except MUX, which has not been tested due to lack of testing equipment; and all modes of RS-232. UPhil can also act as a primary or secondary station.

4.1 Custom Commands with UPhil

When sending custom commands to UPhil, set the baud rate first, followed by the mode.

All commands to UPhil start with `AT\033` and end with `\r`. The format of the command is `AT\033##VV\r`. The `##` is the command number. The `VV` is the value for the command. To read the value of the command, the `VV` is left off and the return value from the read call is the current `VV` value.

4.1.1 Setting baud rate

UPhil runs a 100Mhz clock. To set the baud rate, commands must be sent to the board using the COM AT commands. There are 32 registers on UPhil that can store the baud rate value. The Most Significant Bit (MSB) goes from 31...0. The commands for setting the bits are:

`AT\03330XX\r` is for bits 31..24. (This is for a unbelievably slow baud rate, and is probably never used)

`AT\03331XX\r` is for bits 23..16

`AT\03332XX\r` is for bits 15..8

`AT\03333XX\r` is for bits 7..0

4.1.1.1 Examples for Setting Baud Rate

4.1.1.1.1 Example for DS 101

$100,000,000/64,000 = 1562 = 0x061A$ (Note that the hex value of 1562 is 061A)

`AT\0333206\r` followed by `AT\033331A\r`

4.1.1.1.2 Example RS-232 115,200

$100,000,000/115200 = 868 = 0x0364$

`AT\0333203\r` followed by `AT\0333364\r`

4.1.1.1.3 Example RS-232 9600

$100,000,000/9600 = 10,416 = 0x28B0$

AT\03332**28**\r followed by AT\03333**B0**\r

4.1.1.1.4 Example RS232 2400

$100,000,000 / 2400 = 41666 = 0xA2C2$

AT\03332**A2**\r followed by AT\03333**C2**\r

4.1.1.1.5 2.2.5 Example DS-102

$100,000,000/3000 = 33333.3 = 8235$

AT\03332**82**\r followed by AT\03333**35**\r

4.1.2 Set mode

AT\03301XX is the mode for RS-232/DS-101/DS-102. When setting the mode the shield can be connected between the USB and Fill Port shield for having a continuous ground.

0101 = RS232 (NULL MODEM)
0102 = DS101
0104 = DS102
0108 = RS232
0181 = RS232 (NULL MODEM) with shield pass thru
0182 = DS101 with shield pass thru
0184 = DS102 with shield pass thru
0188 = RS232 with shield pass thru
0180 = ALL OFF with shield pass thru
0100 = ALL OFF

4.1.3 Command For Asynchronous COMS (RS232)

This should not need to be set for normal EKMS operations. However, commands are provided.

4.1.3.1 Stop bits

Number of stop bits (has to be one of these)

AT\03336XX\r where
02 = 1 stop bit (Default)
03 = 1.5 stop bits
04 = 2 stop bits.

4.1.3.2 Parity

AT\03335XX\r

00 = None (Default)
01 = odd
02 = even
03 = mark
04 = space
05 = Parity bit exists but ignore it

4.1.3.3 Data bits

AT\03334XX\r

08 = 8 data bits (Default)
07 = 7 data bits
06 = 6 data bits

4.1.4 Setting LEDs

The LED supports Red, Green and Blue colors. Each color ranges in value from 0 to 255. All three colors need to be set for the color to change to the expected value.

4.1.4.1 Red LED

To modify the red color, issue the command AT\03320XX\r, where XX is a value between 0 (0x00) to 255 (0xFF).

4.1.4.2 Green LED

To modify the green color, issue the command AT\03321XX\r, where XX is a value between 0 (0x00) to 255 (0xFF).

4.1.4.3 Blue LED

To modify the blue color, issue the command AT\03322XX\r, where XX is a value between 0 (0x00) to 255 (0xFF).

4.1.4.3.1 Sample LED Commands

Example – Set the UPhil to red. Issue three commands.

AT\03320FF\r

AT\0332100\r

AT\0332200\r

Example – Set the UPhil to green. Issue three commands.

AT\0332000\r

AT\03321FF\r

AT\0332200\r

Example – Set the UPhil to blue. Issue three commands.

AT\0332000\r

```
AT\0332100\r
```

```
AT\03322FF\r
```

4.1.5 Other commands

4.1.5.1 Flip Polarity on Data bits

It is not recommended to play with this value. In general, this function will not be needed.

```
AT\03337XX\r
```

00 = leave normal

01 = bits will be reversed (and the earth will fly off its axis)

4.1.5.2 Sending DS-101 Wakeup

The user first sets the value to the level for output (0DXX), and then sends the enable command (0EXX).

Sending DS-101 Wakeup is controlled by two registers. Each register controls a different signal. XX is the Logic level to output.

```
AT\0330DXX\r
```

00 = Ground

01 = Logic High = 3.3 volts

Enable the output of the wakeup level

```
AT\0330EXX\r
```

00 = Drives the output level.

01 = Output disabled

Example to send a wakeup

```
AT\0330D01\r followed by AT\0330E00\r
```

4.1.5.3 Receiving DS-101 Wakeup

To get the current set value for the DS-101 wakeup, a command can be sent. The command to get the value is AT\03301\r. The value of the DS-101 will be returned on the buffer. It will need to be read like other data (e.g., ReadFile). The returned values will be:

```
AT\03301\r
```

```
AT\03301XX\r
```

00 = No wakeup

01 = Wakeup

4.1.5.4 Send DS-102 CTS

This is a little different than the DS-101. Only one step is needed. The output enabled is assumed.

AT\03311XX\r
00 = Logic 0. CTS OFF
01 = Logic 1. Physical voltage flip. CTS ON.

4.1.5.5 Receive DS-102 CTS

To get the current set value for the DS-102 CTS, a command can be sent. The command to get the value is AT\03302\r. The value of the CTS will be returned on the buffer. It will need to be read like other data (e.g., ReadFile). The returned values will be:

00 = Logic 0. CTS OFF
01 = Logic 1. Physical voltage flip. CTS ON.

4.1.5.6 Auto Control DS-102 CTS

In EKMS secondary mode, the CTS is turned off by default once the clock is received from the primary station. This feature can be enabled or disabled.

Sending AT\0332300\r will allow for manual control of the CTS levels

Sending AT\0332301\r will allow for automatic operations control of the CTS levels

4.1.5.7 DS-102 CFD Multiplex data

CFD Multiplex data is not decoded for DS-101 on line B.

AT\0330F00\r to signal B (raise voltage)

AT\0330F01\r to disable B

4.1.5.8 DS-102 Bit Inversion (RX/TX)

AT\0331B00\r to set receive (RX) bit order to default DS-102

AT\0331B01\r to set receive (RX) bit order to inverted

AT\0331C00\r to set receive (TX) bit order to default DS-102

AT\0331C01\r to set receive (TX) bit order to inverted

4.1.5.9 Read FPGA Version

This command will read the version of the FPGA.

AT\0333AXX\r

The return version number should be 6.

4.1.5.10 Connect Fill Port and USB shield

To short the Fill Port to USB shield for a continuous ground register bit 6 of register 1 must be set. See section 4.1.2 for Set Mode commands.

4.1.5.11 Active Zeroize Buffers

UPhil will actively zeroize 0.5 seconds after communication. However, to manually perform an active zeroize the following commands can be sent.

To perform a zeroize send the following command. All buffers in the microprocessor and all FIFOs in the are written with 0.

```
AT\0334001\r
```

Once the command is issued, return the UPhil back to the standard state. Issue the following command:

```
AT\0334001\r
```

The command can be written sequentially.

4.1.6 Example Command Sets

4.1.6.1 Example to Set Up Communication Using DS-101

```
AT\0333206\r  
AT\033331A\r  
AT\0330102\r
```

4.1.6.2 Example to Set Up Communication Using DS-101 Wakeup-On

```
AT\0333206\r  
AT\033331A\r  
AT\0330102\r  
AT\0330D00\r  
AT\0330E00\r
```

4.1.6.3 Example to Set Up Communication Using DS-232

```
AT\0333228\r  
AT\0333380\r  
AT\0330101\r
```

4.1.6.4 Example to Set Up Communication Using DS-102

```
AT\0333282\r  
AT\0333335\r  
  
AT\0330104\r
```

4.2 Application Development

4.2.1 RS-232

Once UPhil is running and the COM port is setup, a developer application can talk to UPhil like any other COM port. The application developer should follow the EKMS 308 standards for communicating over RS-232. The communication protocol must be implemented by the developer, and will include items similar to the following:

- a. Allowing 1 or 7 DUs per frame
- b. Responding to an AXID request
- c. Making the frames and escaping the correct bytes

4.2.1.1 RS-232 Sample Code

This will enable RS-232 at 9600 baud and turn on the red LED.

```
// set baud to 9600
string baud1 = "AT\0333364\r";
string baud2 = "AT\03333B0\r";

// send RS232
pPort->Write( (const void*) baud1.c_str(), baud1.length());
pPort->Write( (const void*) baud2.c_str(), baud2.length());

// set the mode
string mode = "AT\0330108\r";
pPort->Write( (const void*) mode.c_str(),mode.length() );

// red at full
string color = "AT\03320FF\r";
pPort->Write( (const void*) color.c_str(), color.length());
color = "AT\0332100\r";
pPort->Write( (const void*) color.c_str(), color.length());
color = "AT\0332200\r";
pPort->Write( (const void*) color.c_str(), color.length());
```

4.2.2 DS-101

DS-101 is a bit level protocol. UPhil will take care of the bit level protocols on behalf of the application. The application would implement writing to the COM port just as it would in the RS-232 case. This would include writing the 7E at the beginning and end of the frame and escaping using 1B.

4.2.2.1 DS-101 Sample code

```
// set baud to 64000 - DS101
string baud1 = "AT\0333206\r";
string baud2 = "AT\033331A\r";

pPort->Write( (const void*) baud1.c_str(), baud1.length());
pPort->Write( (const void*) baud2.c_str(), baud2.length());

// set the mode
string mode = "AT\0330102\r";
pPort->Write( (const void*) mode.c_str(),mode.length() );

// green at full
string color = "AT\0332000\r";
pPort->Write( (const void*) color.c_str(), color.length());
color = "AT\03321FF\r";
pPort->Write( (const void*) color.c_str(), color.length());
color = "AT\0332200\r";
pPort->Write( (const void*) color.c_str(), color.length());
```

4.2.3 DS-102

In DS-102, there are no EKMS packet protocols to implement. Basically, the user opens the COM port and either sends or receives data. It is about as simple as it gets.

4.2.3.1 Sample Code to Send a Test Key via DS-102 with CTS to an ECU

```
void SendTestKeysDS102(HANDLE hPort)
{
    bool b = true;
    unsigned char tStr[256] = {'\0'};
    unsigned char tCTS[4] = {'\0'};
    DWORD Bytes = 0;

    //// DS102
    SendString(hPort, "AT\0333282\r");           // Baud Rate (3K)
    SendString(hPort, "AT\0333300\r");
```

```
SendString(hPort, "AT\0330104\r");    // Set Mode DS102
SendString(hPort, "AT\0332300\r");    // Set CTS control to manual mode

SendString(hPort, "AT\03322FF\r");    // blue
SendString(hPort, "AT\0332000\r");    // red
SendString(hPort, "AT\0332100\r");    // green

while (b)
{
    SendString(hPort, "AT\03302\r");    // Baud Rate (3K)
    memset(tCTS, '\0', 4);
    ReadFile(hPort, tCTS, 4, &Bytes, 0);
    if (tCTS[0] == '\0' && tCTS[1] == '\0' && tCTS[2] == '\r')
    {
        break;
    }
    Sleep(250);
}

tStr[0] = 0x00;
tStr[1] = 0x01;
tStr[2] = 0x02;
tStr[3] = 0x03;
tStr[4] = 0x04;
tStr[5] = 0x05;
tStr[6] = 0x06;
tStr[7] = 0x07;
tStr[8] = 0x08;
tStr[9] = 0x09;
tStr[10] = 0x0A;
tStr[11] = 0x0B;
tStr[12] = 0x0C;
```

```
tStr[13] = 0x0D;
tStr[14] = 0x0E;
tStr[15] = 0x0F;

WriteFile(hPort, tStr, 16, &Bytes, 0);
}
```

4.2.3.2 Sample Code to Receive a Key via DS-102, including a CTS command

// UPhil example to receive a key via DS-102 including a Clear-to-Send command.

```
#include "stdafx.h"
#include "windows.h"
#include <conio.h>
#include <vector>

using namespace std;

HANDLE hPort;

extern void SendString( HANDLE hPort, char *Str )
{
    char tStr[256];
    DWORD Bytes;

    sprintf(tStr, Str);
    WriteFile(hPort, tStr, strlen(tStr), &Bytes, 0);

    Sleep(25);
}

int _tmain(int argc, _TCHAR* argv[])
{
    DWORD err = 0;
```



```
DCB                                PortDCB;

DCB                                m_OriginalDCB;


// Open the port.
if ((hPort = CreateFile(
    L"COM9", GENERIC_READ | GENERIC_WRITE, 0, NULL,
    OPEN_EXISTING, 0, NULL) ) == INVALID_HANDLE_VALUE) {

    err = GetLastError();

    return err;
}


// Get the current device control settings.
if (!GetCommState(hPort, &PortDCB)) {
    err = GetLastError();

    return(err);
}

// *** Configure the port.
// Restore character transmission and place the transmission line in
// a nonbreak state.
if (!EscapeCommFunction(hPort, CLRDTR/*BREAK*/)) {
    err = GetLastError();

    return(err);
}

// Get the current device control settings.
if (!GetCommState(hPort, &PortDCB)) {
    err = GetLastError();

    return(err);
}
```

```
}  
  
// Save the control settings so we restore later.  
m_OriginalDCB = PortDCB;  
  
// Set the new device control block settings.  
PortDCB.DCBlength          = sizeof(DCB);  
  
// PCR #612 [DES] Baud rate must be 9600 for SKL  
PortDCB.BaudRate           = 57600;  
PortDCB.fBinary             = TRUE;  
PortDCB.fParity             = TRUE;  
PortDCB.fOutxCtsFlow        = FALSE;  
PortDCB.fOutxDsrFlow        = FALSE;  
PortDCB.fDtrControl         = DTR_CONTROL_ENABLE;  
PortDCB.fDsrSensitivity     = FALSE;  
PortDCB.fTXContinueOnXoff   = FALSE;  
PortDCB.fOutX               = FALSE;  
PortDCB.fInX               = FALSE;  
PortDCB.fErrorChar          = FALSE;  
PortDCB.fNull               = FALSE;  
PortDCB.fRtsControl         = RTS_CONTROL_ENABLE;  
PortDCB.fAbortOnError       = TRUE;  
PortDCB.ByteSize            = 8;  
PortDCB.Parity              = NOPARITY;  
PortDCB.StopBits            = ONESTOPBIT;  
  
// Init port with new settings.  
if (!SetCommState(hPort, &PortDCB)) {  
    err = GetLastError();  
  
    return(err);  
}
```

```
if (!EscapeCommFunction(hPort, CLRBREAK)) {  
    err = GetLastError();  
  
    return(err);  
}  
  
// set short timeouts on the comm port.  
COMMTIMEOUTS timeouts;  
  
timeouts.ReadIntervalTimeout = 1;  
timeouts.ReadTotalTimeoutMultiplier = 1;  
timeouts.ReadTotalTimeoutConstant = 1;  
timeouts.WriteTotalTimeoutMultiplier = 1;  
timeouts.WriteTotalTimeoutConstant = 1;  
SetCommTimeouts(hPort, &timeouts);  
  
bool b = true;  
unsigned char tStr[256] = {'\0'};  
unsigned char tCTS[4] = {'\0'};  
DWORD Bytes = 0;  
vector<unsigned char> vecKey;  
  
// Set to DS102 mode and CTS to manual mode  
SendString(hPort, "AT\0333282\r");           // Baud Rate (3K)  
SendString(hPort, "AT\0333300\r");  
SendString(hPort, "AT\0330104\r");           // Set Mode DS102  
  
// Turn the LED to blue  
SendString(hPort, "AT\03322FF\r");           // blue  
SendString(hPort, "AT\0332000\r");           // red  
SendString(hPort, "AT\0332100\r");           // green
```

```
// Set CTS control to automatic mode - drops after receiving on clock
SendString(hPort, "AT\0332301\r");

// Just get one key
for (int i = 0; i < 1; ++i)
{
    memset(tStr, '\0', 256);
    vecKey.clear();

    // turn CTS ON
    SendString(hPort, "AT\0331101\r");    // CTS ON

    // read one key
    bool flag = false;
    int flagCount = 0;
    int maxFlagRetries = 5;
    int maxMultipleKeyRetries = 40;
    while (true)
    {
        if (true == flag && flagCount > maxFlagRetries )
        {
            break;
        }
        Sleep(100); // just to keep it from taking too much processor
        ReadFile(hPort, tStr, 256, &Bytes, 0);
        if (Bytes > 0)
        {
            unsigned char* p = &tStr[0];
            printf("%d bytes received\n", Bytes);
            vecKey.insert(vecKey.end(), tStr, p + Bytes);
            flagCount = 0;
            flag = true;
        }
    }
}
```

```
        continue;
    }

    if (true == flag && Bytes == 0)
    {
        flagCount++;
        printf("Key (%d) received of %d bytes.\n", i, vecKey.size());
    }
}

// sleep some time before waiting for next key
Sleep(4000);
}

return(0);
}
```

4.2.3.3 Sample Code to Load and A and B Segment

```
// UPhil example to load an A and B segment into a KIT-1C or KIR-1C.
// KIT-1C is a DS-102 device.
```

```
#include "stdafx.h"
#include "windows.h"
#include <conio.h>

HANDLE hPort;

extern void SendString( HANDLE hPort, char *Str )
{
    char tStr[256];
    DWORD Bytes;

    sprintf(tStr, Str);
```

```
WriteFile(hPort, tStr, strlen(tStr), &Bytes, 0);

Sleep(25);

}

int _tmain(int argc, _TCHAR* argv[])
{
    DWORD err = 0;

    DCB PortDCB;
    DCB m_OriginalDCB;

    // Open the port.
    if ((hPort = CreateFile(
        L"COM9", GENERIC_READ | GENERIC_WRITE, 0, NULL,
        OPEN_EXISTING, 0, NULL) ) == INVALID_HANDLE_VALUE) {

        err = GetLastError();

        return err;
    }

    // Get the current device control settings.
    if (!GetCommState(hPort, &PortDCB)) {
        err = GetLastError();

        return(err);
    }

    // Configure the port.
    // Restore character transmission and place the transmission line in
    // a nonbreak state.
    if (!EscapeCommFunction(hPort, CLRDTR/*BREAK*/)) {
```

```
        err = GetLastError();

        return(err);
    }

    // Get the current device control settings.
    if (!GetCommState(hPort, &PortDCB)) {

        err = GetLastError();

        return(err);
    }

    // Save the control settings so we restore later.
    m_OriginalDCB = PortDCB;

    // Set the new device control block settings.
    PortDCB.DCBlength          = sizeof(DCB);

    // Baud rate must be 9600 for SKL
    PortDCB.BaudRate           = 57600;
    PortDCB.fBinary             = TRUE;
    PortDCB.fParity             = TRUE;
    PortDCB.fOutxCtsFlow       = FALSE;
    PortDCB.fOutxDsrFlow       = FALSE;
    PortDCB.fDtrControl         = DTR_CONTROL_ENABLE;
    PortDCB.fDsrSensitivity     = FALSE;
    PortDCB.fTXContinueOnXoff   = FALSE;
    PortDCB.fOutX               = FALSE;
    PortDCB.fInX               = FALSE;
    PortDCB.fErrorChar          = FALSE;
    PortDCB.fNull               = FALSE;
    PortDCB.fRtsControl         = RTS_CONTROL_ENABLE;
```

```
PortDCB.fAbortOnError          = TRUE;
PortDCB.ByteSize               = 8;
PortDCB.Parity                 = NOPARITY;
PortDCB.StopBits               = ONESTOPBIT;

// Init port with new settings.
if (!SetCommState(hPort, &PortDCB)) {
    err = GetLastError();

    return(err);
}

if (!EscapeCommFunction(hPort, CLRBREAK)) {
    err = GetLastError();

    return(err);
}

// set short timeouts on the comm port.
COMMTIMEOUTS timeouts;

timeouts.ReadIntervalTimeout = 1;
timeouts.ReadTotalTimeoutMultiplier = 1;
timeouts.ReadTotalTimeoutConstant = 1;
timeouts.WriteTotalTimeoutMultiplier = 1;
timeouts.WriteTotalTimeoutConstant = 1;
SetCommTimeouts(hPort, &timeouts);

bool b = true;
unsigned char tStr[256] = {'\0'};
unsigned char tCTS[4] = {'\0'};
DWORD Bytes = 0;
```



```
// Enable DS102 Mode with manual CTS control
SendString(hPort, "AT\0333282\r");      // Baud Rate (3K)
SendString(hPort, "AT\0333300\r");
SendString(hPort, "AT\0330104\r");      // Set Mode DS102
SendString(hPort, "AT\0332300\r");      // Set CTS control to manual mode

// make the led blue
SendString(hPort, "AT\03322FF\r");      // blue
SendString(hPort, "AT\0332000\r");      // red
SendString(hPort, "AT\0332100\r");      // green

SendString(hPort, "AT\0330F01\r");      // Drop B
Sleep (32);
SendString(hPort, "AT\0331101\r");      // CTS ON
Sleep (4);
SendString(hPort, "AT\0331100\r");      // CTS OFF
Sleep (80);
SendString(hPort, "AT\0330F00\r");      // Raise B
Sleep (40);
SendString(hPort, "AT\0331101\r");      // CTS ON
Sleep (200);

tStr[0] = 0x11;
tStr[1] = 0x22;
tStr[2] = 0x33;
tStr[3] = 0x44;
tStr[4] = 0x55;
tStr[5] = 0x66;
tStr[6] = 0x77;
tStr[7] = 0x88;
tStr[8] = 0x99;
```

```
tStr[9] = 0x01;
tStr[10] = 0x02;
tStr[11] = 0x03;
tStr[12] = 0x04;
tStr[13] = 0x05;
tStr[14] = 0x06;
tStr[15] = 0x07;

WriteFile(hPort, tStr, 16, &Bytes, 0);
SendString(hPort, "AT\0331100\r");           // CTS OFF

Sleep(1700);
SendString(hPort, "AT\0330F01\r");           // Drop B
Sleep (32);
SendString(hPort, "AT\0331101\r");           // CTS ON
Sleep (4);
SendString(hPort, "AT\0331100\r");           // CTS OFF
Sleep (80);
SendString(hPort, "AT\0330F00\r");           // Raise B
Sleep (40);
SendString(hPort, "AT\0331101\r");           // CTS ON
Sleep (200);

tStr[0] = 0x20;
tStr[1] = 0x21;
tStr[2] = 0x22;
tStr[3] = 0x23;
tStr[4] = 0x24;
tStr[5] = 0x25;
tStr[6] = 0x26;
tStr[7] = 0x27;
tStr[8] = 0x28;
```

```
tStr[9] = 0x29;  
tStr[10] = 0x30;  
tStr[11] = 0x31;  
tStr[12] = 0x32;  
tStr[13] = 0x33;  
tStr[14] = 0x34;  
tStr[15] = 0x35;  
  
WriteFile(hPort, tStr, 16, &Bytes, 0);  
SendString(hPort, "AT\0331100\r");      // CTS OFF  
  
return(0);  
}
```

5 Linux Programming with UPhil

In the Linux directory, there are two examples of how UPhil can be accessed programmatically. One is using `/dev/ttyACM0` and the other is by talking to the device using `libusb-1.0.0`. The script `<uphil_root>/bin/uphil` is an example of using the `/dev/ttyACM0`. The file `<uphil_root>/src/uphil/rs232.c` contains an example on how to use `libusb` to talk to UPhil directly.

For embedment into key loading devices, it is recommended to use `libusb` for greater control.

Note: The next revision of the firmware will allow separation of the USB interfaces. One interface will be for commands and the other for data.

6 Additional Information

6.1 FAQs

What is the maximum Baud Rate?

We have shown the default signal rates for the EKMS protocols. The UPhil board can run at a higher baud rate. At higher baud rates, some signal roll off may occur. In the lab, we have sent RS-232 working 4Mb/per second. For DS101, we think can go to 500K bits/sec without an issue. The higher rates have not been field tested as of yet.

What is the current?

UPhil draws 165mA of current over the USB port. It can be used on handheld, mobile phones or PCs. Future firmware updates are planned and will lower the current draw by ½ or more.

What is the USB speed?

UPhil runs at USB full speed.

6.2 Support

WebSec can support the integration and development associated with the EKMS protocols and UPhil. We have a mature set of libraries, application and test equipment to support EKMS development for almost any project.

6.3 Data Collection Device (DCD)

The DCD is an EKMS oscilloscope, logic analyzer, and packet analyzer built into one device. It is invaluable for developing new EKMS devices and troubleshooting existing communication issues.

6.4 EKMS Software Libraries

WebSec has over a decade of development working with EKMS software. We have built software libraries that work on Windows, Linux, Android, and the Harris's Sierra II (ARM926-EJS).

For assistance or questions regarding UPhil, please contact David Webster at 858.229.9875 or at websterdav@webseccorp.com.