



UW Script Example Script User Guide

© 2007 COPYRIGHT Ezurio Ltd

This document is issued by Ezurio Limited (hereinafter called Ezurio) in confidence, and is not to be reproduced in whole or in part without the prior written permission of Ezurio. The information contained herein is the property of Ezurio and is to be used only for the purpose for which it is submitted and is not to be released in whole or in part without the prior written permission of Ezurio.

Contents

<u>1.</u>	<u>Introduction</u>	<u>3</u>
	1.1 Use of UwTerminal For Client / Server	3
	1.2 References.....	3
<u>2.</u>	<u>Scripts</u>	<u>4</u>
	2.1 ADC_test.....	4
	2.2 PIO_Test_Output	4
	2.3 PIO_Test_Input.....	4
	2.4 RSSI_Meter.....	5
	2.5 Search_Test2.....	5
	2.6 Tcp_echo	5
	2.7 Test_getipaddr	5
	2.8 bridge_client.....	5
	2.9 bridge_server	8
	2.10 Web_ev_demo.....	10
	2.11 Command_Demo	14
	2.12 Email_Demo	19
	2.13 tftp_client	26

Introduction

1.1 Use of UwTerminal For Client / Server

A number of the test scripts require the use of a TCP client or server to complete the demonstration. The UwTerminal utility can perform either function. The use of the application in this way is described in [1] which is available from Ezurio.

1.2 References

[1] "UwTerminal Users Guide"

2. Scripts

2.1 ADC_test

This script is a simple infinite loop which enables and reads the 3 ADC channels writing the results to the screen.

```
UwTerminal v0.986 Beta
Terminal | Script | Config | About
CTS DSR DCD RI RTS DTR BREAK LocalEcho LineMode Clear
adc1 = 515 adc2 = 466 adc5 = 26
adc1 = 515 adc2 = 466 adc5 = 28
adc1 = 515 adc2 = 466 adc5 = 27
adc1 = 516 adc2 = 466 adc5 = 26
adc1 = 515 adc2 = 467 adc5 = 29
adc1 = 515 adc2 = 466 adc5 = 26
adc1 = 515 adc2 = 467 adc5 = 28
adc1 = 516 adc2 = 467 adc5 = 27
adc1 = 515 adc2 = 466 adc5 = 27
adc1 = 515 adc2 = 467 adc5 = 29
adc1 = 515 adc2 = 466 adc5 = 26
adc1 = 515 adc2 = 467 adc5 = 29
adc1 = 503 adc2 = 466 adc5 = 27
adc1 = 510 adc2 = 466 adc5 = 27
adc1 = 517 adc2 = 467 adc5 = 29
adc1 = 516 adc2 = 466 adc5 = 26
adc1 = 515 adc2 = 467 adc5 = 29
adc1 = 515 adc2 = 466 adc5 = 26
adc1 = 515 adc2 = |
[COM1:115200,N,8,1]
```

adc1 is measuring an internal 1.5V reference and hence should read back roughly 512. adc2 reads the voltage on pin 3 of the dev kit and adc5 reads the voltage applied to pin 1 of the dev kit. This is an infinite loop that must be terminated by resetting the module.

2.2 PIO_Test_Output

Demonstrates using the PIO pins as outputs from the module. Simply sets all PIO signals to be outputs and then steps through each one in turn setting and resetting it. The script waits for 50ms between each set / reset operation. **Note:** GPIO6 is grounded on the current version of the Ezurio development board and so will not toggle.

The script also demonstrates the use of the **pollevent** keyword, by allowing the user to break out of the infinite loop by setting break on the UART Rx input line to the module (selected using the BREAK control in the UWTerminal window).

2.3 PIO_Test_Input

Demonstrates using the PIO pins as inputs. Simply sets all of the PIO pins to be inputs and reads the state of all of the pins and prints the resultant value to the screen. This is an infinite loop that must be terminated by resetting the module.

2.4 RSSI_Meter

Assumes that the module is already attached to an access point (either by using an autorun script or by attaching manually using the **!_wlanattach(apname)** function). The script uses the **_getrssi()** command to report the average SNR measured from the AP. The average SNR is probably the best measure of signal quality returned by the **_getrssi()** command.

2.5 Search_Test2

Performs a search using the **_wlansearchnet()** function and then interrogates the results displaying the results for each available AP in turn. It contains a function that converts the mac address reported by the search command into a more standard format for display. This is a good example of string manipulation in the UW script language.

2.6 Tcp_echo

Demonstrates the creation of a tcp server on port 7. The script assumes that the module is already attached to an access point (either by using an autorun script or by attaching manually using the **!_wlanattach(apname)** function). Before the script is run, the IP address of the module must be identified which is done by using the **!_getmyipaddr\$()** command. When the script it establishes a server on the ip address at port 7 and displays 'waiting for connection...'

The simplest way to connect to the server is to use UWTerminal configured as a client connecting to the module IP address on port 7. When the correct IP address and port are selected, UwTerminal will display the normal terminal screen and the connected indicator will be green. Any characters typed at the terminal screen will be echoed back through the module and displayed.

This script uses the 'long hand' method of establishing a TCP server (i.e it uses the **_socketcreate, _sockbind, _socklisten, _sockaccept**) suite of functions in turn to create the server. A quicker way of achieving the same result is to use the **_socketserver()** function which implements the same functionality in a single call. This is the approach taken in the **bridge_server** script, for example.

2.7 Test_getipaddr

Assumes that the module is already attached to an access point (either by using an autorun script or by attaching manually using the **!_wlanattach(apname)** function). Uses **_getmyipaddr\$()** and **_getmybrdaddr\$()** to display the IP and broadcast addresses provided by DHCP, then it deletes this address and replaces it using the **_myipaddrset()** function. Finally it displays the modified IP addresses.

2.8 bridge_client

This script demonstrates the transfer of data between the UART and an open socket using the data bridge mechanism. This is the method of data transfer that is most likely to be used for cable replacement / data transfer applications as it eliminates the script from the data path and allows it to reach its full throughput.

```
//  
// First define some global variables.  
//  
slong sock  
slong grc  
uword guwrc  
ulong gepuart  
ulong geptcp  
ulong gepstd
```

```

string gserver
//
// Define where to find the server. This is likely to change.
//
gserver = "10.0.0.101:19"
//
// Subroutine to attach to AP and wait for IP address to be allocated.
//
function uword Attach_Wlan(string byval stSsid)
    uword uwRes
    string stipadd

    uwRes = _wlanattach(stSsid)
    if (uwRes == 0) then
        print "Waiting for DHCP"
        stipadd = _getmyipaddr$()
        while strlen(stipadd) == 0
            sleep(500)
            print "."
            stipadd = _getmyipaddr$()
        endwhile
        print "\n"
        print "my ip address = "; stipadd ; "\n"
    else
        print "Failed to attach to "; stSsid; "\n"
    endif
endfunc uwRes
//
// Assert subroutine simplifies error handling in the rest of the script.
//
subroutine assert(slong val)
    slong dummy
    if val == 0 then
        print "ASSERT:", val, "\n"
        dummy = val/0
    endif
endsub
//
// Subroutine to attach to server. Returns when the socket is connected.
// Uses the netio function to assign a bridge endpoint to the socket.
//
function along connect(string server)
    along endpoint
    slong rc

    rc = _socketpcreate(sock)
    if rc then
        exitfunc 0
    endif

    rc = _sockconnect(sock, server)
    if rc then
        rc = _sockclose(sock)
        exitfunc 0
    endif

    endpoint = _netio(sock)
    if endpoint == 0 then
        rc = _sockclose(sock)
        exitfunc 0
    endif
endfunc endpoint
//
// This subroutine is called if the socket close event is detected. It closes the socket
// and breaks the connection between the socket and UART and re-connects the UART to the
// UWScript.
//
function uword close_handler()
    slong rc

    rc = _sockclose(sock)
    unbridge(1)
    guwrc = bridge(2, gepstd, gepuart)
    if guwrc != 0 then
        assert(0)
    endif
endfunc 0
//

```

```

// This subroutine is called if the break event is detected. It closes the socket
// and breaks the connection between the socket and UART and re-connects the UART to the
// UWscript. This provides a mechanism to break the UART / socket connection from outside.
//
function uword brkon_handler()
    slong slRes

    slRes = _sockclose(sock)
    unbridge(1)
    guwrc = bridge(2, gepstd, gepuart)
    if guwrc != 0 then
        assert(0)
    endif
endfunc 0
//
// Main body of the script.
//
onevent evsocketclose call close_handler
onevent evrxbrkon    call brkon_handler

if (Attach_Wlan("timslinksys") == 0) then

    geptcp = connect(gserver)
    if geptcp == H'FFFFFFFF then
        assert(0)
    endif

    gepuart = uartio(0)
    if gepuart == H'FFFFFFFF then
        assert(0)
    endif

    gepstd = stdio()
    if gepstd == H'FFFFFFFF then
        assert(0)
    endif

    unbridge(0)

    guwrc = bridge(1, geptcp, gepuart)
    if guwrc != 0 then
        assert(0)
    endif
    waitevent
endif

```

In addition to bridging the script introduces a number of concepts:

- Use of an ASSERT subroutine as a common handler for all errors generated in the script. The assert function forces a break from the script by generating a divide by zero error.
- Waiting for the **_getmyipaddr\$()** to return a non-empty string as a way of detecting when DHCP has provided an IP address. Currently this script waits forever – a better approach might be to implement a timeout.
- Use of event handlers to catch asynchronous events and handle them appropriately. Two events are supported – **evsocketclose** which is triggered by the far end closing the socket and **evrxbrkon** which is triggered when a break condition is detected on the UART Rx line. Both of these events close the socket and re-connect the UART to UWscript prior to forcing a script exit.

To demonstrate the script, first configure it for your environment, this includes changing the server IP address and port number (if required) – these are both set at the top of the script. Also, the AP SSID should be changed, this is defined by the call to Attach_Wlan call which is in the main body of the script.

Once the script has been customised and downloaded, set up a server using UWTerminal, at the port number set in the script. Then simply run the script, and it will connect to the AP, wait for and IP address from DHCP and then connect to the server. While the module is connected data typed at either end will appear at the other (via the operation of the bridge).

The module will remain connected to the server until either the server disconnects or a break is generated on the UART Rx line (using the BREAK checkbox in UwTerminal) at which point the script will clean up and exit. The module will now be back in interactive mode (remember to uncheck the BREAK checkbox once the script has exited).

2.9 bridge_server

This script demonstrates the transfer of data between the UART and an open socket using the data bridge mechanism. This is the method of data transfer that is most likely to be used for cable replacement / data transfer applications as it eliminates the script from the data path and allows it to reach its full throughput.

The script has many of the same concepts as the bridge_client script and differs only in as much as it opens a server connection and waits for a client rather than connecting to a server. It uses the **_socketpserver()** high level function that performs all of the socket creation, bind and connect functionality. This function returns when a client has connected, otherwise it blocks.

To demonstrate the script, first configure it for your environment by changing the AP SSID, which is defined by the call to Attach_Wlan call which is in the main body of the script at the end of the file.

Once the modified script is downloaded and run, it will attach to the access point and then display the IP address allocated by DHCP. Setup a client using UwTerminal using the module IP address and port number 58594, this should connect immediately to the module and bridging should be enabled.

The module will remain connected to the client until either the client disconnects or a break is generated on the UART Rx line (using the BREAK checkbox in UwTerminal) at which point the script will clean up and exit. The module will now be back in interactive mode (remember to uncheck the BREAK checkbox once the script has exited).

```

// Establishes server socket and bridges to socket when client connects.
// RESTRICTION: This server can only handle one client at a time!
//
// First define some global variables.
//
    ulong gepuart
    ulong geptcp
    ulong gepstd
    uword guwrc
    slong sILstnSock
    slong sIDataSock
//
// This subroutine is called if the socket close event is detected. It closes the socket
// and breaks the connection between the socket and UART and re-connects the UART to the
// UWscript.
//
function uword close_handler()
    slong rc

    rc = _sockclose(sIDataSock)
    unbridge(1)
    guwrc = bridge(2, gepstd, gepuart)
endfunc 0
//
// This subroutine is called if the break event is detected. It closes the socket
// and breaks the connection between the socket and UART and re-connects the UART to the
// UWscript. This provides a mechanism to break the UART / socket connection from outside.
//
function uword brkon_handler()
    slong sIRes

    sIRes = _sockclose(sIDatasock)
    unbridge(1)
    guwrc = bridge(2, gepstd, gepuart)
endfunc 0
//
// Subroutine to attach to AP and wait for IP address to be allocated.
//
function uword Attach_Wlan(string byval stSsid)
    uword uwRes
    string stipadd

    uwRes = _wlanattach(stSsid)
    if (uwRes == 0) then
        print "Waiting for DHCP"
        stipadd = _getmyipaddr$()
        while strlen(stipadd) == 0
            sleep(500)
            print "."
            stipadd = _getmyipaddr$()
        endwhile
        print "\n"
        print "my ip address = "; stipadd ; "\n"
    else
        print "Failed to attach to "; stSsid; "\n"
    endif
endfunc uwRes
//
// Subroutine to set up the server and wait for the client to connect.
//
subroutine EchoServer()
    slong sIRes
    string stLocalName
    string stRecvBuf
    slong finish
    string stMyip

    print "my ip address = "; _getmyipaddr$(); "\n"

    sIRes = _socktcpserver(sIDataSock, ":58594")

    print "Established connection\n"
    gepuart = uartio(0)
    gepstd = stdio()
    geptcp = _netio(sIDataSock)
    unbridge(0)
    guwrc = bridge(1, geptcp, gepuart)
endsub

```

```
//  
// Main body of the script.  
//  
onevent evsocketclose call close_handler  
onevent evrxbrkon call brkon_handler  
  
if (Attach_Wlan("timslinksys") == 0) then  
    EchoServer()  
    Waitevent  
endif
```

2.10 Web_ev_demo

This script and the associated web page files provides a demonstration of the web page integration possible from within the module. Since there are a number of files associated with this demo, the complete web demo is in the 'Web demo' subdirectory.

In addition to downloading the Web_ev_demo script, the associated web page files should be downloaded using the Download – Data File+ option in UwTerminal (see Ref [1]). The following files must be downloaded in this way:

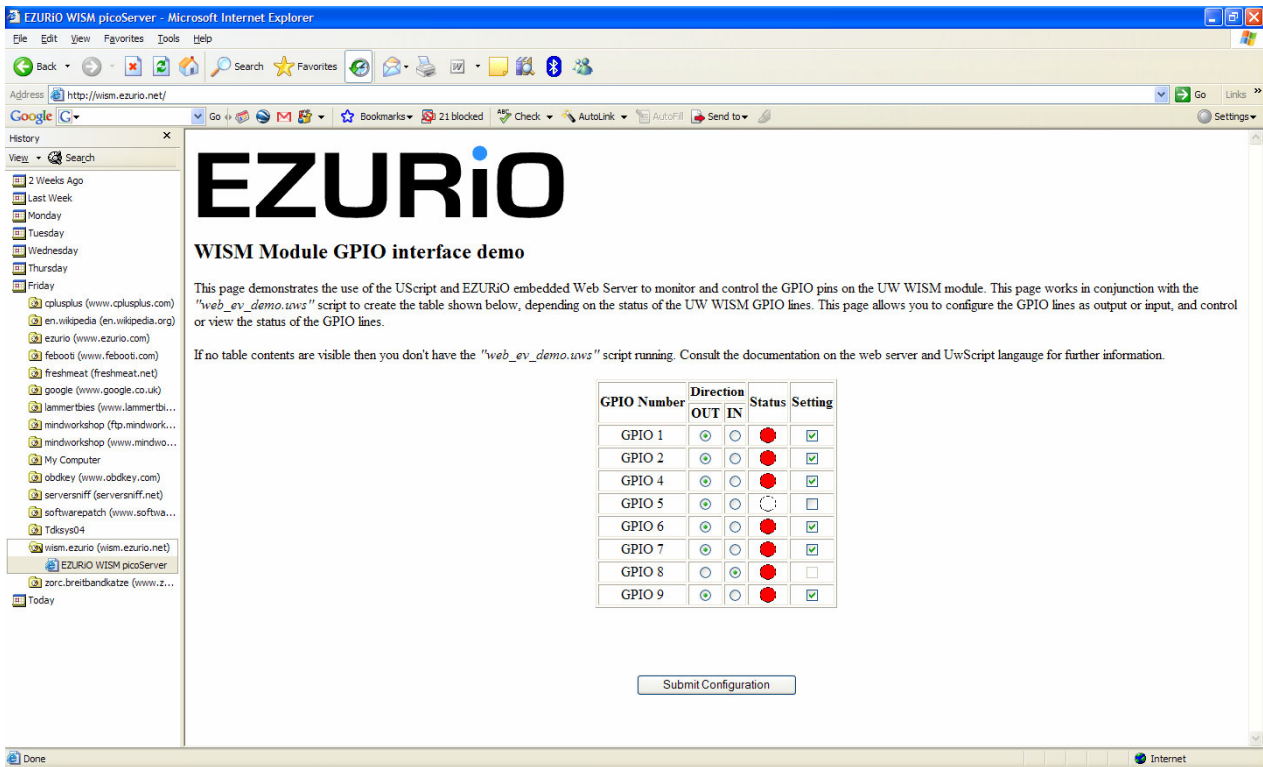
```
Index.html  
Colorless_led.png  
Red_led.png  
Ezurio_96pt  
Refresh.html  
stopScript.html
```

The script provides a simple user interface to control the GPIO lines on the module – it allows the GPIO direction to be set and then the output state to be defined or the input state to be displayed.

In implementing this functionality, this script / web page combination demonstrates all of the supported features of the WISM web integration namely:

- Provision of dynamic web page content from a script. When constructs are encountered in the html file, the script is called to deliver dynamic web content to the browser.
- Support of web browser posts. When information is posted from the browser (such as when the state of a GPIO line is changed) then the script is also called to intercept the posted information and act on it as appropriate.

Once all of the files have been downloaded, run the Web_ev_demo script. The module web page can be accessed in the normal way by entering the IP address of the module into the URL field of a standard browser. The web page displayed is shown below:



The current state of the GPIO signals is displayed via dynamic web page content and is controlled by the Gpio_Get_Status subroutine. This subroutine is called by the main web event handler, webhandler, after the cause of the **EVWEB** event has been determined to be a dynamic web content tag embedded in the web page by the **_webEvGet()** in built function.

If the configuration of the GPIO signals is modified from within the browser and submitted, then the new configuration is posted to the module using an HTTP POST which triggers the **EVWEB** event.


```

scriptname = LEFT$(tag, UWCASTSL(charPos))

'// then get the args - in this case there can be only one
startPos = UWCASTSL(charPos)
findChar = ")"
charPos = STRPOS( tag, findChar, startPos )
arg1 = MID$( tag, startPos+1, (UWCASTSL(charPos) - startPos - 1) )

if mStrCompare(scriptName, "GpioGetStatus") == 0 then
    GpioGetStatus( UWCASTSL(STRVALDEC(arg1)) )
    sRes = _WebEvEnd()
    exitfunc 1
elseif mStrCompare(scriptName, "ExitScript") == 0 then
    sRes = _WebEvEnd()
    exitfunc 0
endif

elseif ( evCause == 2 ) then
    numElements = _webGetNumPostEl()
    for idx=0 to numElements
        '// get each element and carry out required action
        result = _webGetElement(idx, name, value)

        '// find "gpiodirX" entity
        findChar = "dir"
        charPos = STRPOS( name, findChar, 0 )
        if ( charPos != -1 ) then
            '// extract the GPIO number
            arg1 = RIGHT$(name, 1)
            if ( mStrCompare(value, "output") == 0 ) then
                _GPIOSETDIR( UWCASTSL(STRVALDEC(arg1)), 0 )
            else
                _GPIOSETDIR( UWCASTSL(STRVALDEC(arg1)), 1 )
            endif
        endif

        '// find "gpiostateX" entity
        findChar = "state"
        charPos = STRPOS( name, findChar, 0 )
        if ( charPos != -1 ) then
            '// extract the GPIO number
            arg1 = RIGHT$(name, 1)
            if ( mStrCompare(value, "on") == 0 ) then
                _GPIOSETPIN( UWCASTSL(STRVALDEC(arg1)) )
            endif
        else
            _GPIOCLRPIN( UWCASTSL(STRVALDEC(arg1)) )
        endif

        '// find "toggleOutputs" entity if it has been sent
        findChar = "toggleOutputs"
        charPos = STRPOS( name, findChar, 0 )
        if ( charPos != -1 ) then
            if ( mStrCompare(value, "toggleYes") == 0 ) then
                print "We should be toggling now, but I haven't added the script code yet\n"
            endif
        endif
    next

    '// load the refresh page
    ulFile=fopen("refresh.html",0)
    if (ulfile) then
        _webechostart()
        do
            result=fread(ulFile,arg1,500)
            webecho arg1
        until result == 0

        '// close the file
        result=fclose(ulFile)
    endif
    '// we have finished so end the web event
    sRes = _WebEvEnd()
    exitfunc 1
else
    exitfunc 1
endif
endfunc 1

```

```
print "UwScript Web Event handler is starting.\n"
```

```
ONEVENT EVWEB call webhandler  
// now we wait for the event handler to kick  
waitevent
```

2.11 Command_Demo

The Command_Demo script demonstrates the creation of a custom user interface for the module. The script introduces some new concepts, namely:

- Use of the **evstdin** event that is generated when any input arrives on stdin. By default when a script is started, stdin is bridged to the UART and so this event is generated if any input arrives on the UART. However, if stdin is bridged to any other device (an open socket for example), then the event will be generated if any data arrives on that device.
- Extracting command lines from stdin using
- Creation of a command look up table using the **TableInit()** and **TableAdd(...)** built-in functions.
- Extracting data from stdin using the **stdin()** built-in function, checking for line terminators using **strpos()** and then extracting completed lines using the **left\$()** function and **strrotleft()** functions.
- Using non-volatile user variable storage to store and recall entered parameters over power cycles.
- Use of the **evlink** event to handle changes in the wireless LAN link status and the **_wlanlinkstatus()** built-in function to identify the cause of the change in link state (loss of AP connection for example).

The script also contains some useful functions and subroutines that can be used in other scripts that are creating command line interpreters:

- **Extract-Token()** – Extracts the next token from a command line. The token de-limiter is specified as an input parameter. The input command line is modified so that repeated calls to **Extract-Token** will extract all of the tokens included on a command line.
- **Process_Line()** – Parses a command line. Takes the first token on the command line and checks it against the command look up table and then uses a **select** statement to implement the functionality associated with each command token.
- **Skipwhite()** – Skips over non-printing characters in a string (space, tab etc). Useful for removing redundant spaces etc in command lines.

The script splits into three main sections:

- Connection related code. This contains routines that attach to the WLAN AP and establish the server on the specified port. This section also contains the event handlers for handling socket closure and loss of the AP.
- Command handler code. This contains all of the functionality to handle incoming data on stdin, detect completed command lines and then parsing keywords against expected table entries to identify what function is required.
- Main program section. This sets up key global variables (such as the prompt string, command line terminators and keyword delimiters), sets up the command look up table and then simply sits on **waitevent** waiting for commands to be typed.

Once the script has been downloaded and run, the command handler prompt WISM> will be displayed. The following commands can be used:

- ssid ssidname – Specifies the SSID of the AP to use.
- com :nnn – Port number to use for the server.
- key keystring – specifies the WEP key to be used.
- list – Displays the current settings.
- store – stores the current settings in the non volatile RAM
- load – loads the settings from non-volatile RAM
- connect – connects to the AP and sets up a TCP/IP server on the specified port.
- search- performs an AP search and displays the results.
- quit – Quits the command line interface script.

The following screenshot shows a typical usage of the command interface script. In this case Hyperterminal was used to establish the client connection to 10.0.0.102:7.

```

UwTerminal v0.986 Beta
Terminal | Script | Config | About |
CTS DSR DCD RI RTS DTR BREAK LocalEcho LineMode Clear

WLAN>search
MAC=00:18:39:8D:78:3A TYPE=2 FREQ=246200000 SEC=D800 RSSI=216 SSID="timslinksys"
MAC=00:18:3F:20:A4:19 TYPE=2 FREQ=243700000 SEC=A700 RSSI=167 SSID="Ezurio Wi-Fi"
MAC=00:11:50:92:57:0E TYPE=2 FREQ=241200000 SEC=C700 RSSI=199 SSID="dimitris"
MAC=00:0E:84:4B:0D:B6 TYPE=2 FREQ=241200000 SEC=A500 RSSI=165 SSID="VOCUK_GUEST"
MAC=00:18:39:6A:38:52 TYPE=2 FREQ=242700000 SEC=C500 RSSI=197 SSID="Ezurio"
WLAN>ssid timslinksys
ssid = timslinksys
WLAN>com :7
com = :7
WLAN>list
timslinksys, , :7
WLAN>store
WLAN>load
WLAN>list
timslinksys, , :7
WLAN>connect
Connecting to timslinksys
Open attach
Waiting for DHCP..
my ip address = 10.0.0.102
Waiting for client to connect on:7
Established connection
WLAN>Socket disconnect

00
[COM1:115200,N,8,1]

```

```

// Script to implement a simple customer user interface for the
// module. The script allows the user to configure ssid, key and port
// number before creating a server and waiting for a connection.
//
// Command handler related variables

```

```

string stRxBuf
uword uwRxLen
string stLineDelim
string stTokDelim
string stToken
string stLine
string stTable
sword swRes
sword swIndex
string stSSID
string stkey
string stcom

```

```

string stPrompt

//=====
//=====
//=====
// WLAN / TCP Connection section
//=====
//=====
//=====
// Connection related variables

ulong gepuart
ulong geptcp
ulong gepstd
uword guwrc
slong sILstnSock
slong sIDataSock

function uword link_handler()
uword uwReturn

if _wlanlinkstatus() == 0 then
unbridge(1)
guwrc = bridge(0, gepstd, gepuart)
print "Lost AP\n"
uwReturn = 0
else
uwReturn = 1
endif
endfunc uwReturn

function uword close_handler()
slong rc

' sleep(5000)
rc = _sockclose(sIDataSock)
unbridge(1)
guwrc = bridge(0, gepstd, gepuart)
print "Socket disconnect\n"
endfunc 0

subroutine EchoServer()
slong sIRes
string stipadd

if (_wlanlinkstatus() == 1) then
print "Disconnecting existing connection\n"
_wlandetach()
_myipaddrdel()
endif
print "Connecting to "; stSSID; "\n"
if strlen(stkey) != 0 then
print "Secure attach "; stkey; strlen(stkey); "\n"
_wlankey(stkey)
_wlansecurity(1)
else
print "Open attach\n"
_wlansecurity(0)
endif
if _wlanattach(stSSID) != 0 then
print "Failed to attach to "; stSSID; "\n"
else
print "Waiting for DHCP"
stipadd = _getmyipaddr$( )
while strlen(stipadd) == 0
sleep(500)
print "."
stipadd = _getmyipaddr$( )
endwhile
print "\n"
print "my ip address = "; stipadd ; "\n"
print "Waiting for client to connect on" ; stcom; "\n"
sIRes = _socktcpserver(sIDataSock, stcom)
print "Established connection\n"
gepuart = uartio(0)
gepstd = stdio()
geptcp = _netio(sIDataSock)

```

```

        unbridge(0)
        guwrc = bridge(1, geptcp, gepuart)
    endif
endsub

'//=====
'//=====
'//=====
'// Command Handler section
'//=====
'//=====
'//=====
'//=====

subroutine AddTable(string byref strTble, string byval strTok, uword index)
swRes=TableAdd(strTble,strTok,index)
if swRes!=0 then
    print "\nTableAdd() failed"
endif
endsub

subroutine skipwhite(string byref stInput)
sword swNextChar
uword exit
uword index

exit = 0
index = 0

' print "H3\n"
if strlen(stInput) != 0 then
    while exit == 0
        swNextChar = strgetchr(stInput, 0)
        if swNextChar <= H'20 then
            strtroleft(stInput, 1)
            if strlen(stInput) == 0 then
                exit = 1
            endif
        else
            exit = 1
        endif
    endwhile
endif
endsub

function string extract_token(string byref stInput, string byval stTokens)
string stReturn
slong slPos
uword uwPos
uword index
uword match
string stNextChar
uword uwstlen

index = 0
match = 0
stReturn = " "
uwstlen = strlen(stInput)
while (index != uwstlen) && (match == 0)
    stNextChar = MID$(stInput, index, 1)
    slPos=strpos(stTokens, stNextChar, 0)
    if slPos >= 0 then
        match = 1
        stReturn = left$(stInput, index)
        strtroleft(stInput, index)
    else
        index = index+1
    endif
endwhile
if (index == uwstlen) then
    stReturn = stInput
    strtroleft(stInput, index)
endif
endfunc stReturn

'//=====
'// This subroutine is called to parse an input line
'//=====

```

```

function uword process_line(string byref stLine)
    uword uwPos
    string stNextToken
    string stSpace
    uword uwReturn
    sword swReturn

    uwReturn = 1
    stSpace= " "
    skipwhite(stLine)
    if (strlen(stLine) != 0) then
        stNextToken = extract_token(stLine, stTokDelim)
        skipwhite(stLine)
        if (strcmp(stNextToken, stSpace) != 0) then
            swIndex=TableLookup(stTable,stNextToken)
            select swIndex
            case 0
                stNextToken = extract_token(stLine, stTokDelim)
                stSSID = stNextToken
                print "ssid = "; stNextToken; "\n"
            case 1
                stNextToken = extract_token(stLine, stTokDelim)
                stkey = stNextToken
                print "key = "; stNextToken; "\n"
            case 2
                stNextToken = extract_token(stLine, stTokDelim)
                stCom = stNextToken
                print "com = "; stNextToken; "\n"
            case 3
                print stSSID; ", "; stkey; ", "; stCom; "\n"
            case 4
                Echoserver()
            case 5
                _wlansearchdump()
            case 6
                print "ssid nn: Enters SSID\n"
                print "key nn: Enters WEP key\n"
                print "com nn: Enters port for socket connection (must have : as first character)\n"
                print "list: Lists current parameters\n"
                print "search: Performs and displays available access points\n"
                print "help: Displays this list\n"
            case 7
                swReturn = nvrecordset(1, stSSID)
                swReturn = nvrecordset(2, stCom)
                swReturn = nvrecordset(3, stkey)
            case 8
                if nvrecordget(1, stSSID) == 0 then
                    stSSID="3Com"
                endif
                if nvrecordget(2, stCom) == 0 then
                    stCom=":58594"
                endif
                if nvrecordget(3, stkey) == 0 then
                    stkey=""
                endif
            case 100
                print "\nQUITting application...\n"
                uwReturn = 0
            case else
                print "\nToken unknown\n"
            endselect
        endif
    endif
    print stPrompt
endfunc uwReturn

//=====================================================
// This handler is called when new data in stdin has arrived
//=====================================================
function uword stdinhandler()
    uword uwPos
    slong slPos
    uword uwReturn

    uwReturn = 1
    //Read all from stdin stream into our string variable
    uwRxLen=stdin(stRxBuf)

```

```

//Check if the line delimiter exists in what we have so far
slPos=strpos(stRxBuf,stLineDelim,0)
' print stRxBuf; " "; slPos; "\n"
if slPos >= 0 then
    //at least one line has been detected
    uwPos = uwcastsl(slPos)

    //strip all up to that token and put into a seperate variable
    stLine = left$(stRxBuf,uwPos)

    //rotate our buffer so that the token extracted is no longer there
    strtolleft(stRxBuf, (uwPos + strlen(stLineDelim)))

    //Finally process the Line
    uwReturn = process_line(stLine)
endif
endfunc uwReturn

//=====
//=====
//=====
// Main Program section
//=====
//=====
//=====
// Initialise variables
//=====
stLineDelim="\r"
stTokDelim=" \r\n\t"
stSSID="3Com"
stkey=""
stCom=":58594"
stPrompt="WLAN>"
//=====
// Create lookup table
//=====
swRes=TableInit(stTable)
if swRes!=0 then
    print "\nTableInit() failed"
endif

AddTable(stTable,"ssid",0)
AddTable(stTable,"key",1)
AddTable(stTable,"com",2)
AddTable(stTable,"list",3)
AddTable(stTable,"connect",4)
AddTable(stTable,"search",5)
AddTable(stTable,"help",6)
AddTable(stTable,"store",7)
AddTable(stTable,"load",8)
AddTable(stTable,"quit",100)

//-----
// Enable synchronous event handlers
//-----
OnEvent EVSTDIN call stdinhandler
onEvent EVLINK call link_handler
onevent evsocketclose call close_handler

//-----
// Wait for a synchronous event.
// A Script can have multiple <WaitEvent> statements
//-----
print stPrompt
'while 1
    WaitEvent
'endwhile

```

2.12 Email_Demo

The email_demo script is an extension to the command_demo script, adding an SMTP client protocol to allow the user to send simple e-mails to a known server. The script adds the following additional commands to the user interface developed in command_demo:

- serverip ipaddress – Sets the IP address of a server
- client – Establishes a client connection to serverip:com
- email – Starts the e-mail transaction.

For an e-mail transaction the port number should be set to 25 (i.e COM :25). When the e-mail transaction is started, the script requests the TO and FROM addresses and the subject for the e-mail before connecting to the AP and the server and starting the SMTP protocol. The screenshot below shows typical use.

Within the script the SMTP_Open and SMTP_Close subroutines are responsible for the initial negotiation and closure of the SMTP session.

```

UwTerminal v0.986 Beta
Terminal | Script | Config | About
CTS [x] DSR [x] DCD [x] RI [x] RTS [x] DTR [x] BREAK [ ] LocalEcho [x] LineMode [ ] Clear
WLAN>serverip
server =
WLAN>ssid Ezurio
ssid = Ezurio
WLAN>com :25
com = :25
WLAN>email
TO>tim.wheatley@ezurio.com
FROM>tim
SUBJECT>Test
Connecting to Ezurio
Open attach
Waiting for DHCP.....
my ip address = 192.168.1.150
Establishing connection to server
Established connection
220 tdksys01.www.tdksys.com Microsoft ESMTMP MAIL Service, Version:
5.0.2195.6713 ready at Tue, 27 Feb 2007 08:58:30 +0000
250 tdksys01.www.tdksys.com Hello
MAIL FROM: tim
250 2.1.0 tim@tdksys.com...Sender OK
RCPT TO: tim.wheatley@ezurio.com
250 2.1.5 tim.wheatley@ezurio.com
DATA
354 Start mail input; end with <CRLF>.<CRLF>
From: <tim>
To: <tim.wheatley@ezurio.com>
Subject: Test
EMAIL>This is a test message.
EMAIL>This is the second time.
EMAIL>Thanks
EMAIL>
.
250 2.6.0 <TDKSYS01z6BJ8QiIWMz0000078b@tdksys01.www.tdksys.com> Queued mail
for delivery
QUIT
221 2.0.0 tdksys01.www.tdksys.com Service closing transmission channel
WISM>
[COM1:115200,N,8,1]

```

```

// Script to implement a simple customer user interface for the
// module. The script allows the user to configure ssid, key and port
// number before creating a server and waiting for a connection. In addition
// the script allows e-mails to be constructed and sent.
//
// Command handler related variables

string stRxBuf
uword uwRxLen
string stLineDelim
string stTokDelim
string stToken
string stLine
string stTable
sword swRes
sword swIndex
string stSSID
string stkey
string stcom
string stPrompt
string stserver
string stTo
string stFrom
string stSubject
uword uwState
string stEmailTerm

//=====
//=====
//=====
// WLAN / TCP Connection section
//=====
//=====
//=====
// Connection related variables

ulong gepuart
ulong geptcp
ulong gepstd
uword guwrc
slong slStnSock
slong slDataSock

function uword link_handler()
uword uwReturn

if _wlanlinkstatus() == 0 then
unbridge(1)
guwrc = bridge(0, gepstd, gepuart)
print "Lost AP\n"
uwReturn = 0
else
uwReturn = 1
endif
endfunc uwReturn

function uword close_handler()
slong rc

sleep(5000)
rc = _sockclose(slDataSock)
unbridge(1)
guwrc = bridge(0, gepstd, gepuart)
print "Lost socket\n"
endfunc 0

subroutine EchoServer(uword client_server)
slong slRes
string stipadd

if (_wlanlinkstatus() == 1) then
print "Disconnecting existing connection\n"
_wlandetach()
_myipaddrdel()
endif
print "Connecting to "; stSSID; "\n"
if strlen(stkey) != 0 then
print "Secure attach "; stkey; strlen(stkey); "\n"

```

```

        _wlankey(stkey)
        _wlansecurity(1)
    else
        print "Open attach\n"
        _wlansecurity(0)
    endif
if _wlanattach(stSSID) != 0 then
    print "Failed to attach to "; stSSID; "\n"
else
    print "Waiting for DHCP"
    stipadd = _getmyipaddr$()
    while strlen(stipadd) == 0
        sleep(500)
        print "."
    stipadd = _getmyipaddr$()
    endwhile
    print "\n"
    print "my ip address = "; stipadd ; "\n"
    if (client_server == 0) then
        print "Waiting for client to connect on" ; stcom; "\n"
        slRes = _socketserver(slDataSock, stcom)
        print "Established connection\n"
    else
        print "Establishing connection to server" ; stserver; "\n"
        slRes = _socketclient(slDataSock, stserver+stcom)
        print "Established connection\n"
    endif
    gepuart = uartio(0)
    gepstd = stdio()
    geptcp = _netio(slDataSock)
    unbridge(0)
    guwrc = bridge(1, geptcp, gepuart)
endif
endsub

subroutine SMTP_Open()
    slong slRes
    string stipadd
    string stPrint
    string stInput

    if (_wlanlinkstatus() == 1) then
        print "Disconnecting existing connection\n"
        _wlandetach()
        _myipaddrdel()
    endif
    print "Connecting to "; stSSID; "\n"
    if strlen(stkey) != 0 then
        print "Secure attach "; stkey; strlen(stkey); "\n"
        _wlankey(stkey)
        _wlansecurity(1)
    else
        print "Open attach\n"
        _wlansecurity(0)
    endif
    if _wlanattach(stSSID) != 0 then
        print "Failed to attach to "; stSSID; "\n"
    else
        print "Waiting for DHCP"
        stipadd = _getmyipaddr$()
        while strlen(stipadd) == 0
            sleep(500)
            print "."
        stipadd = _getmyipaddr$()
        endwhile
        print "\n"
        print "my ip address = "; stipadd ; "\n"
        print "Establishing connection to server" ; stserver; "\n"
        slRes = _socketclient(slDataSock, stserver+stcom)
        print "Established connection\n"
        stPrint = "HELO mine\r\n"
        slRes = _socksendall(slDataSock, stPrint, 0)
        sleep(2000)
        slRes = _sockrecv(slDataSock, stInput, 0, 250)
        print stInput
        stPrint = "MAIL FROM: " + stFrom + "\r\n"
        print stPrint
        slRes = _socksendall(slDataSock, stPrint, 0)
    endif
end subroutine

```

```

sleep(2000)
sIRes = _sockrecv(sIDataSock, stInput, 0, 250)
print stInput
stPrint = "RCPT TO: " + stTo + "\r\n"
print stPrint
sIRes = _socksendall(sIDataSock, stPrint, 0)
sleep(2000)
sIRes = _sockrecv(sIDataSock, stInput, 0, 250)
print stInput
stPrint = "DATA\r\n"
print stPrint
sIRes = _socksendall(sIDataSock, stPrint, 0)
sleep(2000)
sIRes = _sockrecv(sIDataSock, stInput, 0, 250)
print stInput
stPrint = "From: <" + stFrom + ">\r\n"
print stPrint
sIRes = _socksendall(sIDataSock, stPrint, 0)
stPrint = "To: <" + stTo + ">\r\n"
print stPrint
sIRes = _socksendall(sIDataSock, stPrint, 0)
stPrint = "Subject: " + stSubject + "\r\n"
print stPrint
sIRes = _socksendall(sIDataSock, stPrint, 0)
endif
endsub

subroutine SMTP_Close()
slong sIRes
string stPrint
string stInput

stPrint = ".\r\n"
print stPrint
sIRes = _socksendall(sIDataSock, stPrint, 0)
sleep(2000)
sIRes = _sockrecv(sIDataSock, stInput, 0, 250)
print stInput
stPrint = "QUIT\r\n"
print stPrint
sIRes = _socksendall(sIDataSock, stPrint, 0)
sleep(2000)
sIRes = _sockrecv(sIDataSock, stInput, 0, 250)
print stInput
endsub

subroutine SMTP_Send(string stInput)
slong sIRes

stInput = stInput + "\r\n"
sIRes = _socksendall(sIDataSock, stInput, 0)
endsub
'//=====
'//=====
'//=====
'// Command Handler section
'//=====
'//=====
'//=====
'//=====

subroutine AddTable(string byref strTble, string byval strTok, uword index)
swRes=TableAdd(strTble,strTok,index)
if swRes!=0 then
print "\nTableAdd() failed"
endif
endsub

subroutine skipwhite(string byref stInput)
sword swNextChar
uword exit
uword index

exit = 0
index = 0

' print "H3\n"
if strlen(stInput) != 0 then

```

```

while exit == 0
  swNextChar = strgetchr(stInput, 0)
  if swNextChar <= H'20 then
    strtrotleft(stInput, 1)
    if strlen(stInput) == 0 then
      exit = 1
    endif
  else
    exit = 1
  endif
endwhile
endif
endsub

```

```

function string extract_token(string byref stInput, string byval stTokens)

```

```

  string stReturn
  slong slPos
  uword uwPos
  uword index
  uword match
  string stNextChar
  uword uwstlen

```

```

  index = 0
  match = 0
  stReturn = " "
  uwstlen = strlen(stInput)
  while (index != uwstlen) && (match == 0)
    stNextChar = MID$(stInput, index, 1)
    slPos=strpos(stTokens, stNextChar, 0)
    if slPos >= 0 then
      match = 1
      stReturn = left$(stInput, index)
      strtrotleft(stInput, index)
    else
      index = index+1
    endif
  endwhile
  if (index == uwstlen) then
    stReturn = stInput
    strtrotleft(stInput, index)
  endif
endfunc stReturn

```

```

'//=====
'// This subroutine is called to parse an input line
'//=====

```

```

function uword process_line(string byref stLine)

```

```

  uword uwPos
  string stNextToken
  string stSpace
  uword uwReturn

```

```

  uwReturn = 1
  select uwState
  case 0
    stSpace= " "
    skipwhite(stLine)
    if (strlen(stLine) != 0) then
      stNextToken = extract_token(stLine, stTokDelim)
      skipwhite(stLine)
      if (strcmp(stNextToken, stSpace) != 0) then
        swIndex=TableLookup(stTable,stNextToken)
        select swIndex
        case 0
          stNextToken = extract_token(stLine, stTokDelim)
          stSSID = stNextToken
          print "ssid = "; stNextToken; "\n"
        case 1
          stNextToken = extract_token(stLine, stTokDelim)
          stkey = stNextToken
          print "key = "; stNextToken; "\n"
        case 2
          stNextToken = extract_token(stLine, stTokDelim)
          stCom = stNextToken
          print "com = "; stNextToken; "\n"
        case 3
          print stSSID; ", "; stkey; ", "; stCom; ", "; stserver; "\n"

```

```

case 4
    Echoserver(0)
case 5
    _wlansearchdump()
case 6
    print "ssid nn: Enters SSID\n"
    print "key nn: Enters WEP key\n"
    print "com nn: Enters port for socket connection (must have : as first character)\n"
    print "list: Lists current parameters\n"
    print "search: Performs and displays available access points\n"
    print "server: Establishes a server connection\n"
    print "client: Establishes a client connection\n"
    print "help: Displays this list\n"
case 7
    Echoserver(1)
case 8
    stNextToken = extract_token(stLine, stTokDelim)
    stserver = stNextToken
    print "server = "; stNextToken; "\n"
case 9
    stPrompt = "TO>"
    uwState = 1
case 100
    print "\nQUITting application...\n"
    uwReturn = 0
case else
    print "\nToken unknown\n"
endselect
endif
endif
case 1
    stTo = stLine
    uwState = 2
    stPrompt = "FROM>"
case 2
    stFrom = stLine
    uwState = 3
    stPrompt = "SUBJECT>"
case 3
    stSubject = stLine
    uwState = 4
    SMTP_Open()
    stPrompt = "EMAIL>"
case 4
    if (strpos(stLine, stEmailTerm, 0) >= 0) then
        SMTP_Close()
        stPrompt = "WISM>"
        uwState = 0
    else
        SMTP_Send(stLine)
    endif
endif
case else
    uwState = 0
    stPrompt = "WISM>"
endselect
print stPrompt
endfunc uwReturn
//=====================================================
// This handler is called when new data in stdin has arrived
//=====================================================
function uword stinhandler()
    uword uwPos
    slong slPos
    uword uwReturn

    uwReturn = 1
    //Read all from stdin stream into our string variable
    uwRxBuf=stdin(stRxBuf)

    //Check if the line delimiter exists in what we have so far
    slPos=strpos(stRxBuf,stLineDelim,0)
    print stRxBuf; " "; slPos; "\n"
    if slPos >= 0 then
        //at least one line has been detected
        uwPos = uwcastsl(slPos)

        //strip all up to that token and put into a separate variable
        stLine = left$(stRxBuf,uwPos)

```

```

//rotate our buffer so that the token extracted is no longer there
strrotleft(stRxBuf, (uwPos + strlen(stLineDelim)))

//Finally process the Line
uwReturn = process_line(stLine)
endif
endfunc uwReturn

//=====
//=====
//=====
// Main Program section
//=====
//=====
//=====
// Initialise variables
//=====
stLineDelim="r"
stTokDelim=" \r\n\t"
stSSID="3Com"
stkey=""
stCom=":58594"
stserver="192.168.1.1"
stPrompt="WLAN>"
stTo="tim.wheatley@ezurio.com"
stFrom="tim.wheatley@ezurio.com"
uwState = 0
stEmailTerm = "\07"
//=====
// Create lookup table
//=====
swRes=TableInit(stTable)
if swRes!=0 then
    print "\nTableInit() failed"
endif

AddTable(stTable,"ssid",0)
AddTable(stTable,"key",1)
AddTable(stTable,"com",2)
AddTable(stTable,"list",3)
AddTable(stTable,"server",4)
AddTable(stTable,"search",5)
AddTable(stTable,"help",6)
AddTable(stTable,"client",7)
AddTable(stTable,"serverip",8)
AddTable(stTable,"email",9)
AddTable(stTable,"quit",100)

//-----
// Enable synchronous event handlers
//-----
OnEvent EVSTDIN call stdinhandler
onEvent EVLINK call link_handler
onevent evsocketclose call close_handler

//-----
// Wait for a synchronous event.
// A Script can have multiple <WaitEvent> statements
//-----
print stPrompt
WaitEvent

```

2.13 tftp_client

This script implements a tftp client protocol. The subroutines tftp_send and tftp_receive send and receive named files to a tftp server. The main section of the program simply calls tftp_send to send the file index.html from the module file system to the server and then calls tftp_receive to get a file from the server. The received file is streamed out of the UART.

To use this script, a tftp client must be available and running on a networked PC. In addition the index.html file must be loaded into the module file system The server IP address which is defaulted in the script to 10.0.0.101 must be modified to match the IP address of the server

machine. Once this is done and the script downloaded, the simply run the script and the file transfers will be completed.

```
// Simple TFTP client implementation
//
slong sISock
string stServerip
string stAddrPort
uword uwblock
string stblock

subroutine Connect_Wlan(string byval stSSID)
    uword uwRes
    string stipadd

    uwRes = _wlanattach(stSSID)

    if (uwRes != 0) then
        print "Failed to attach to network """"; stSSID; """"\n"
    else
        print "Waiting for DHCP"
        stipadd = _getmyipaddr$( )
        while strlen(stipadd) == 0
            sleep(500)
            print "."
            stipadd = _getmyipaddr$( )
        endwhile
        print "\n"
        print "my ip address = "; stipadd ; "\n"
    endif
endsub

subroutine Connect_Server()
    slong sIRes

    print stAddrPort

    ' create the socket
    sIRes = _sockudpcreate(sISock, 0)
    if sIRes != 0 then
        print "Failed to create socket: sIRes="; sIRes; "\n"
        exitsub
    endif

    print "connecting to "; stAddrPort; "\n"
    sIRes = _sockbind(sISock, ":5000")
    ' sIRes = _sockconnect(sISock, stAddrPort)
    if sIRes != 0 then
        print "Failed to connect to "; stAddrPort; "\n"
        sIRes = _sockclose(sISock)
        exitsub
    else
        sIRes = _socknonblock(sISock)
        print "Connected\n"
    endif
endsub

subroutine Close_Server()
    slong sIRes

    sIRes = _sockclose(sISock)
    if sIRes != 0 then
        print "close of listening socket failed: sIRes="; sIRes; "\n"
        exitsub
    endif
endsub

subroutine Send_RRQ(string stFilename)
    string stPacket
    slong sIRes

    stPacket = "\00\01"+stFilename+"\00netascii\00"

    sIRes = _socksendto(sISock, stPacket, 0, stAddrPort)
endsub

subroutine Send_WRQ(string stFilename)
    string stPacket
```

```

slong slRes

stPacket = "\00\02"+stFilename+"\00netascii\00"

slRes = _socksendto(slSock, stPacket, 0, stAddrPort)
endsub

subroutine Send_Ack(uword uwpnum)
string stPacket
slong slRes
sword swRes
uword uwchar

stPacket = "\00\04 "

uwchar = uwpnum / 256
swRes = strsetchr(stPacket, uwchar, 2)
uwchar = uwpnum - (uwchar * 256)
swRes = strsetchr(stPacket, uwchar, 3)

slRes = _socksendto(slSock, stPacket, 0, stAddrPort)
endsub

subroutine Send_Error(uword uwenum, string stError)
string stPacket
slong slRes
sword swRes
uword uwchar

stPacket = "\00\05 " + stError + "\00"

uwchar = uwenum / 256
swRes = strsetchr(stPacket, uwchar, 2)
uwchar = uwenum - (uwchar * 256)
swRes = strsetchr(stPacket, uwchar, 3)

slRes = _socksendto(slSock, stPacket, 0, stAddrPort)
endsub

subroutine Send_Data(uword uwblocknum, string stData)
string stPacket
slong slRes
sword swRes
uword uwchar

stPacket = "\00\03 " + stData

uwchar = uwblocknum / 256
swRes = strsetchr(stPacket, uwchar, 2)
uwchar = uwblocknum - (uwchar * 256)
swRes = strsetchr(stPacket, uwchar, 3)

slRes = _socksendto(slSock, stPacket, 0, stAddrPort)
' print stPacket; "\n"
endsub
'//
'// Check_Reponse: Checks a response packet from the server. Returns the following:
'// 0: Error or unexpected response. Abort the session
'// 1: Acknowledgement is OK, proceed
'// 2: Response timeout
'//
function uword Get_Response(uword uwSequence)
sword swOpcode
uword uwReturn
string stError
sword swValue
slong slRes
uword timer
string stInput
string stOption
slong slNullpos
string stCompare
uword uwTemp

timer = 0
uwReturn = H'FFFF
while (uwReturn == H'FFFF)
slRes = _sockrecvfrom(slSock, stInput, 0, 600, stAddrPort)

```

```

' print "Socket Rx returned "; sIRes; "\n"
while ((sIRes == d'11) && (sIcastuw(timer) < d'2000))
  sIRes = _sockrecvfrom(sISock, stInput, 0, 600, stAddrPort)
' print "Socket Rx returned "; sIRes; "\n"
  timer = timer + 1
endwhile
if (timer >= 2000) then
  print "Timeout...\n"
  uwReturn = 2
else
  swOpcode = strgetchr(stInput, 0) * 256
  swOpcode = swOpcode + strgetchr(stInput, 1)
  select swOpcode
  case 4
    swValue = strgetchr(stInput, 2) * 256
    swValue = swValue + strgetchr(stInput, 3)
    if swvalue == swcastuw(uwSequence) then
      print "Acknowledgment OK\n"
      uwReturn = 1
    endif
  case 5
    swValue = strgetchr(stInput, 2) * 256
    swValue = swValue + strgetchr(stInput, 3)
    print "Error Code "; swValue; " "
    strrotleft(stInput, 4)
    print stInput; "\n"
    uwReturn = 0
  case 6
    stCompare = "\00"
    sINullpos = strpos(stInput, stCompare, 1)
    uwTemp = uwcastsl(sINullPos - 2)
    stOption = MID$(stInput, 2, uwTemp)
    print stOption; "\n"
    strrotleft(stInput, uwcastsl(sINullpos+1))
    print stInput; "\n"
    stCompare = "blksize"
    if strcmp(stOption, stCompare) == 0 then
      stCompare = "\00"
      sINullpos = strpos(stInput, stCompare, 0)
      uwTemp = uwcastsl(sINullPos - 1)
      stOption = MID$(stInput, 0, uwcastsl(sINullpos - 1))
      print stOption; "\n"
      sIRes = strvaldec(stOption)
      uwblock = uwcastsl(sIRes)
      uwReturn = 1
    else
      print "Did not understand option ack\n"
      uwReturn = 0
    endif
  case else
    print "Unexpected Opcode "; swOpcode; "\n"
    uwReturn = 0
  endselect
endif
endwhile
endfunc uwReturn
'//
'// Get_Data: Gets the next data packet from the server. If the next packet from the server is not
'// a data packet then the function returns a non-zero value.
'//
function uword Get_Data(string stPacket, uword byref uwSequence)
  sword swOpcode
  uword uwReturn
  string stError
  sword swValue
  slong sIRes
  uword timer
  string stInput

  timer = 0
  uwReturn = H'FFFF
  while (uwReturn == H'FFFF)
    sIRes = _sockrecvfrom(sISock, stPacket, 0, 600, stAddrPort)
    ' print "Socket Rx returned "; sIRes; "\n"
    while ((sIRes == d'11) && (sIcastuw(timer) < d'2000))
      sIRes = _sockrecvfrom(sISock, stPacket, 0, 600, stAddrPort)
      ' print "Socket Rx returned "; sIRes; "\n"
      timer = timer + 1
    endwhile
  endwhile
endwhile
endfunc

```

```

endwhile
if (timer >= 2000) then
    print "Timeout...\n"
    uwReturn = 2
else
    swOpcode = strgetchr(stPacket, 0) * 256
    swOpcode = swOpcode + strgetchr(stPacket, 1)
    select swOpcode
    case 3
        swValue = strgetchr(stPacket, 2) * 256
        swValue = swValue + strgetchr(stPacket, 3)
        uwSequence = uwcastsw(swValue)
        strrotleft(stPacket, 4)
        uwReturn = 0
    case 5
        swValue = strgetchr(stInput, 2) * 256
        swValue = swValue + strgetchr(stInput, 3)
        print "Error Code "; swValue; " "
        strrotleft(stInput, 4)
        print stInput; "\n"
        uwReturn = 1
    case else
        print "Unexpected Opcode "; swOpcode; "\n"
        uwReturn = 2
    endselect
endif
endwhile
endfunc uwReturn

```

```

subroutine tftp_Send(string stFilename)
    ulong ulPfile
    string stFstring
    string stResponse
    uword uwSequence
    uword status
    uword uwFinished
    uword uwLength
    uword uwResponse
    uword uwSendInProgress

    ulPfile = fopen(stFilename, 0)
    uwSequence = 0
    uwFinished = 0
    stAddrPort = stServerip + ":69"
    Connect_Server()
    Send_WRQ(stFilename)
    if (Get_Response(uwSequence) != 1) then
        uwFinished = 1
    endif
    while (uwFinished == 0)
        uwSequence = uwSequence + 1
        uwLength = fread(ulPfile, stFstring, uwblock)
        if (uwLength < uwblock) then
            uwFinished = 1
        endif
        uwSendInProgress = 1
        while uwSendInProgress == 1
            Send_Data(uwSequence, stFstring)
            uwResponse = Get_Response(uwSequence)
            select uwResponse
            case 0
                uwFinished = 1
                uwSendInProgress = 0
                print "Error occurred, aborting\n"
            case 1
                uwSendInProgress = 0
            case 2
                print "Re-transmitting "; uwSequence; "\n"
            case else
                print "Unexpected return\n"
                uwSendInProgress = 0
            endselect
        endwhile
    endwhile
endsub

```

```

subroutine tftp_Receive(string stFilename)
    ulong ulPfile

```

```

string stPacket
string stResponse
uword uwSequence
uword status
uword uwFinished
uword uwLength
uword uwResponse
uword uwSendInProgress

uwSequence = 0
uwFinished = 0
stAddrPort = stServerip + ":69"
Connect_Server()
Send_RRQ(stFilename)
while (uwFinished == 0)
  if (Get_Data(stPacket, uwSequence) == 0) then
    if (strlen(stPacket) != uwblock) then
      uwFinished = 1
    endif
    print stPacket
    Send_Ack(uwSequence)
  endif
endwhile
endsub

string stfilename

stServerip = "10.0.0.101"
stfilename = "index.html"
Connect_Wlan("timslinksys")
uwblock = 512
stblock = "512"

tftp_Send(stfilename)
Close_Server()
stfilename = "Test.s"

tftp_Receive(stfilename)
Close_Server()

```