

EZURiO

EZURiO

UWScript Wireless LAN Extensions 2.0.1.1

The information contained in this document is subject to change without notice. EZURiO makes no warranty of any kind with regard to this material including, but not limited to, the implied warranties of merchant ability and fitness for a particular purpose. EZURiO shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

© Copyright 2007 EZURiO Limited.

All rights reserved.

This document contains information that is protected by copyright. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of EZURiO.

Other product or company names used in this publication are for identification purposes only and may be trademarks of their respective owners.

Contents

1.	INTRODUCTION	5
1.2	Modes of Operation	5
1.3	Streams	6
1.4	Event Handling	6
2.	WIRELESS LAN SPECIFIC COMMANDS	7
1.1	_WLANAUTH	7
1.2	_WLANSECURITY	7
1.3	_WLANKEY	7
1.4	_WLANATTACH	7
1.5	_WLANDETACH	7
1.6	_DHCPGETSTATE	7
1.7	_PSMODESET	7
1.8	_PSWKUPSET	7
1.9	_PSWKUPCAUSE	7
1.10	_RSSIGET	7
1.12	_WLANCHANSET	8
1.13	_WLANBSSTYPE	8
1.14	_WLANANTSET	8
1.15	_WLANSEARCHDUMP	8
1.16	_WLANSEARCHNET	8
1.17	_SEARCHCOUNT	8
1.18	_SEARCHQUERY	8
1.19	_SEARCHQUERY\$	9
1.20	_SEARCHQUERYFOR	9
1.21	BRIDGE	9
1.22	UNBRIDGE	9
1.23	STDIO	9
1.24	UARTIO	10
1.25	_NETIO	10
1.26	_UARTMODIFY	10
1.27	_UARTBREAK	10

1.28	<code>_ADCOPEN</code>	10
1.29	<code>_ADCREAD</code>	10
1.30	<code>_GPIOSETPIN</code>	10
1.31	<code>_GPIOCLRPIN</code>	10
1.32	<code>_GPIOSETALL</code>	10
1.33	<code>_GPIOCLRALL</code>	11
1.34	<code>_GPIOSETDIR</code>	11
1.35	<code>_GPIOGETPIN</code>	11
1.36	<code>_GPIOGETALL</code>	11
1.37	<code>_SOCKBIND</code>	11
1.38	<code>_SOCKCONNECT</code>	11
1.39	<code>_SOCKLISTEN</code>	11
1.40	<code>_SOCKACCEPT</code>	11
1.41	<code>_SOCKCLOSE</code>	12
1.42	<code>_SOCKSENDALL</code>	12
1.43	<code>_SOCKSENDTO</code>	12
1.44	<code>_SOCKRECV</code>	12
1.45	<code>_SOCKRECVFROM</code>	12
1.46	<code>_SOCKTCPSERVER</code>	13
1.47	<code>_SOCKTCPCLIENT</code>	13
1.48	<code>_SOCKGETNAME</code>	13
1.49	<code>_SOCKNONBLOCK</code>	13
1.50	<code>_SOCKTCPCREATE</code>	14
1.51	<code>_SOCKUDPCREATE</code>	14
1.53	<code>_GETMACADDR</code>	14
1.54	<code>_GETMYIPADDR\$</code>	14
1.55	<code>_GETMYBRDADDR\$</code>	14
1.57	<code>_MYIPADDRGET</code>	14
1.58	<code>_MYBCASTADDRGET</code>	15
1.59	<code>_DHCPSETSTATE</code>	15
1.60	<code>_MYIPADDRSET</code>	15
1.61	<code>_MYIPADDRDEL</code>	15

1.62	_DNSCONFGET	15
1.63	_DNSCONFSET	15
1.64	FOPEN	15
1.65	FCLOSE	15
1.66	FREAD	15
1.67	_FLASHDEL	15
1.68	ONEVENT	16
2.0	TCP/IP ERROR CODES	17
	APPENDIX 1 - SAMPLE SCRIPTS	19
3.	SIMPLE SCRIPT TO REPORT AP RSSI	22
4.	SIMPLE SCRIPT TO MONITOR ALL GPIOS	23
5.	SIMPLE SCRIPT TO TOGGLE THROUGH ALL GPIOS	23
6.	WEB EVENT HANDELING	25
7.	HTTP GET	28

1. INTRODUCTION

The contents of this document are an extension to the UWScript core language and must be used in context with this original document. This document describes all the additional language commands available to support development of Wireless LAN applications using TCP/IP.

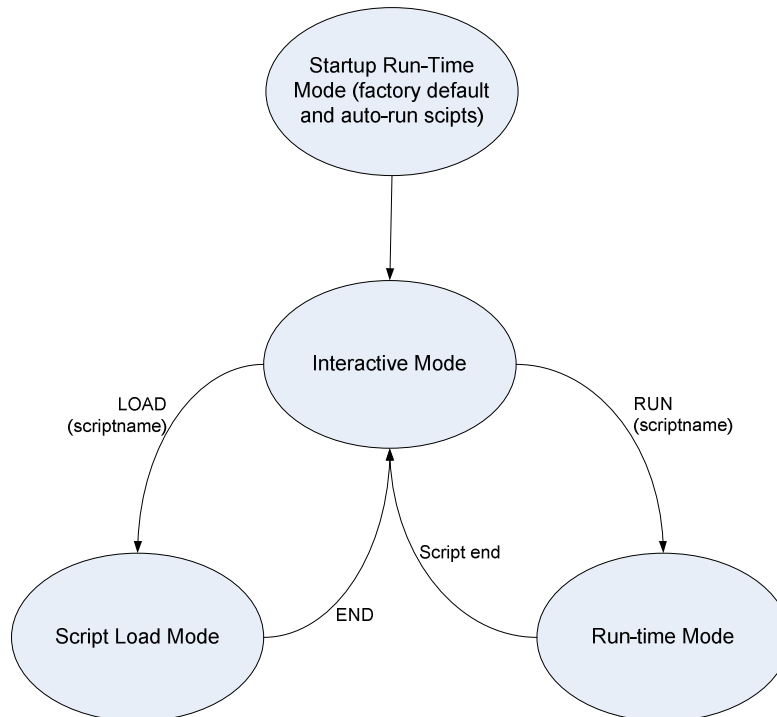
1.2 Modes of Operation

A Wireless LAN module has three modes of operation:

- Interactive mode. Commands sent via the UART are interpreted and executed immediately. Interactive mode can be used to configure and control the module directly via the UART using UWScript commands. Interactive mode is also used to manage the download and storage of scripts that will be used later in run-time mode.
- Script load mode. The module switches into script load mode when the LOAD(scriptname) command is sent by the host. In script load mode, all UWScript commands received are checked for syntax correctness, tokenised (to minimise storage requirements) and then stored in non-volatile memory as part of the named script.
- Run-time mode. Pre-stored command scripts are read from non-volatile memory and executed.

By default the module starts up in run-time mode and executes the factory default and auto-run scripts (described below). On completion of these scripts (provided that they do complete) the module switches to interactive mode and waits for commands to be received over the UART.

The mode of the module changes depending on commands from the host according to the following state diagram:



The ability to execute pre-stored scripts allows UWScript to provide a wireless independent interface to the host software. For example, the host software may simply execute an 'establish_connection' run-time script which will configure the module to send data. The pre-stored script may be very different between Bluetooth and WiFi modules, but the host command is the same (RUN establish_connection) and the result will be the same. This allows the same host software to be used independent of the underlying wireless technology – all of which is handled by the pre-stored script.

1.3 Streams

TCP/IP uses Streams as a mechanism by which data is transferred between data sources and sinks within a UW module.

Each potential source of data within the system has an associated stream (for example the UART, TCP/IP socket, file system, UW interpreter etc) and any data generated by the source is placed into the stream. Each potential sink of data within the system (for example UART, TCP/IP socket etc) can be connected to one or more streams via UW commands. When a sink of data is connected to a stream, it is notified when data is placed in the stream and can then retrieve the data for its own use.

For example, in a simple cable replacement application, the UART is the data source and the TCP/IP socket is the data sink (and vice-versa). Data is placed into the UART stream by the UART interrupt handler and is retrieved from the stream by the TCP/IP task – when sufficient data has been retrieved by the TCP/IP task it is placed into a packet and sent via the TCP/IP socket.

Certain data sources (for example the UART) and the associated stream are created by default. However, more generally, streams are created when the data source is created. For example, when a TCP/IP socket is created (as a result of the MAKECONNECT command) then the associated stream is created at the same time.

The UW BRIDGE command is used to link a data sink to a particular stream.

1.4 Event Handling

The UWScript language interpreter allows for the occurrence of asynchronous events within a UWScript module. An example of an asynchronous event might be an unexpected loss of connection or the completion of a connection.

Asynchronous events are analogous to interrupts and are handled by event handler routines within the UWScript interpreter. UWScript commands are provided to allow event handler routines to be installed for each possible event source. The list of possible events will change dependent on the wireless technology.

Event handler functions. If the event handler returns 1 then the handler will return to the same point in the script. If the event handler returns 0 then it returns to the first construct in the script.

2. WIRELESS LAN SPECIFIC COMMANDS

1.1 **_WLANAUTH** (UWORD flag)

flag := 0 indicates that shared key authentication is disabled, 1 indicates that shared key authentication is enabled.

Selects the use of shared key authentication over the Wireless Lan connection to the access point. If shared key authentication is selected, then this must be combined with setting the WEP key using the KEY() command.

1.2 **_WLANSECURITY** (UWORD level)

level := 0 for None, 1 for WEP

Selects the encryption method to be applied to the Wireless Lan link. If WEP encryption is selected then this must be combined with setting the WEP key using the KEY() command.

1.3 **_WLANKEY** (STRING key)

key := string representation of the security key Sets the key to be used by the WEP algorithm for encryption.

1.4 **UWORD _WLANATTACH** (STRING strvar)

STRING strvar := Reference to a string variable which specifies the SSID name

This is a builtin function.

Attach to a wireless network.

Returns 0 if could not attach, 1 if it could

1.5 **_WLANDETACH** ()

Detach from the access point.

1.6 **UWORD _DHCPGETSTATE** ()

Returns 1 if the wireless interface is setup to use DHCP, 0 otherwise

1.7 **_PSMODESET** ()

Set the powersave mode of the module.

1.8 **_PSWKUPSET** (UWORD type)

UWORD type := bit mask defining the GPIO lines to monitor when processor is asleep The mask MUST be none zero or the call will fail.

This is a builtin function. It is used to configure the reason for the processor to wakeup when in slow clock sleep mode. This function MUST be called before the PS6 sleep request. Failure to do this will result in the PS6 request returning a fail command. A change in state long enough to trigger a wake condition, on any monitored GPIO pin while the processor is asleep will wake the processor up.

Returns nothing

1.9 **UWORD _PSWKUPCAUSE** ()

This is a builtin function. It is used to get the reason for the processor waking up. The possible reasons for the wakeup event are configured by the _PSWKUPSET subroutine.

Returns The cause of the wakeup event

1.10 **_RSSIGET** (UWORD AvgSnr, UWORD InstSnr, UWORD AvgNf, UWORD InstNf)

AvgSnr := average SNR from beacon

InstSnr := instantaneous SNR from beacon
AvgNf := average noise floor from beacon
InstNf := instantaneous noise floor from beacon Get the RSSI values.

The values are returned through the subroutine arguments which are passed by reference.

1.11 **_WLANREGIONSET** (UWORD region)

region := H'10 for US
 H'20 for Canada
 H'30 for EU
 H'31 for SPAIN
 H'32 for FRANCE
 H'40 for JAPAN

Sets the geographical region in which the module we operate.

1.12 **_WLANCHANSET** (UWORD channel)

channel := Channel number
Selects the channel to be used for an ad-hoc connection.

This command has no meaning in an access point connection (the channel is set by the AP).

1.13 **_WLANBSSTYPE** (UWORD bsstype)

bsstype := 0 for access point
 1 for ad-hoc

Selects whether an ad-hoc (point to point) connection is to be established or an access point connection.

1.14 **_WLANANTSET** (UWORD dir, UWORD mode)

dir := 0 for RX
 1 for TX
Mode:= 1 for internal
 2 for external,
 H'FFFF for diversity

Sets the RX or TX RF antenna mode.

Examples: ANTENNASET(0,1) set the RX antenna to internal antenna
 ANTENNASET(0,2) set the RX antenna to external antenna
 ANTENNASET(1,H'FFFF) set the TX antenna to diversity mode

1.15 **_WLANSEARCHDUMP** ()

Performs a blocking search and dumps the results to the UART.
This function is meant to be used from immediate mode for troubleshooting purposes.

1.16 **_WLANSEARCHNET** (UWORD flags)

flags := set bit 0 to request blocking operation, in blocking mode the command will block until the search data has been gathered. In blocking mode, the OnSearch event isn't raised.

Initiates a wireless network search operation. By default, the command works in non-blocking mode. When it completed it will raise the event OnSearch.

1.17 **UWORD _SEARCHCOUNT** ()

Returns the number of search results found.

1.18 **ULONG _SEARCHQUERY** (UWORD row, UWORD column)

row := selects the results row to work with

column := selects the datum to return
Return a single integer datum from the search result.
The datum is identified by the row number and its column number. Both rows and columns start counting from 0.

The following columns are defined:

- 0 : type, infrastructure or ad-hoc mode
- 1 : frequency, in Mhz
- 2 : security,
- 3 : RSSI

1.19 **STRING _SEARCHQUERY\$** (UWORD row, UWORD column)

row:= selects the results row to work with

column := selects the datum to return

Return a single string datum from the search result.

The datum is identified by the row number and its column number. Both rows and columns start counting from 0.

The following columns are defined:

- 0 : SSID, name of the network found
- 1 : MAC, MAC address of the access point, it is returned as a binary data

1.20 **SWORD _SEARCHQUERYFOR** (STRING name, UWORD startfrom)

name:= SSID to search for

startfrom := row to start from, if the same SSID exists multiple times, then this can be used to skip over already searched results.

Typically you'll set this to 0. Search the results database for a network with the an SSID of 'name' and return its row number.

If a match doesn't exist, then the return will be a negative number. If the same SSID occurs multiple times the first occurrence is returned.

Use the 'startfrom' parameter to look for other occurrences, if needed.

1.21 **UWORD BRIDGE** (UWORD bridgeNum, ULONG SrcEndpoint, ULONG DstEndpoint)

UWORD bridgeNum := A 0 based value which identifies the bridge to setup.

ULONG SrcEndpoint := A Handle to an endpoint which will be source of the data

ULONG DstEndpoint := A Handle to an endpoint which will be the consumer of the data

This is a builtin function.

Returns 0 if bridge could successfully be established. Otherwise it will be a resultcode

The XxxEndpoints are obtained using appropriate functions The following Bridge Numbers are assumed to exist and setup when a script starts running:

- 0 := Script STDIN taking data from UART
- 1 := UART taking and transmitting data from Script STDOUT

1.22 **UNBRIDGE** (UWORD bridgeNum)

UWORD bridgeNum := A 0 based value which identifies the bridge to tear down.

This is a builtin subroutine.

1.23 **ULONG STDIO** ()

This is a builtin function.

Returns 0 if endpoint is invalid, otherwise a handle which identifies the scripting entity as a consumer and source of streaming data.

For example, any PRINT statements and output of immediate commands are written to STDOUT which can be patched through to any reader.

1.24 ULONG UARTIO (UWORD UartPortNumber)

UWORD UartPortNumber := A 0 based value which identifies the physical uart port.

This is a builtin function.

Returns 0 if endpoint is invalid, otherwise a handle which identifies the uart port specified as a streaming endpoint

1.25 ULONG _NETIO (SLONG sockid)

socketid := the socket ID

Binds a TCP socket with the an endpoint and returns the endpoint handle. The connection becomes live immediately. Any data coming from the network, are given to the endpoint. Any data passed to the endpoint are given to the network socket.

The binding is destroyed when:

- a) a network error occurs on the socket
- b) the network socket is closed down (by either the server or the client)

1.26 _UARTMODIFY (ULONG baud, UWORD bits, UWORD parity, UWORD stop, UWORD persist)

Sets the UART port parameters.

1.27 _UARTBREAK (UWORD state)

state := set state to 1, to set the UART BREAK condition. Set state to 0, to remove the UART BREAK condition

Enables/disables the UART BREAK condition.

1.28 _ADCOPEN (UWORD channel)

channel := channel number to initialise, valid values: 0,1 and 5.

Initialises a ADC channel so that it can be read with ADCREAD.

1.29 ULONG _ADCREAD (UWORD channel)

channel := channel number to access

valid values: 0,1 and 5.

Returns the value of the ADC channel.

1.30 _GPIOSETPIN (UWORD pin)

UWORD pin := pin to set, Valid pin numbers are: 1,2,4,5,6,7,8,9.

Set the value of the GPIO pin.

1.31 _GPIOCLRPN (UWORD pin)

UWORD pin := pin to clear, Valid pin numbers are: 1,2,4,5,6,7,8,9.

Clear the value of the GPIO pin.

1.32 _GPIOSETALL ()

Set GPIO pins.

Any pin that is assigned the value 1 is set. Pins that are given the value 0 are left as they are.

1.33 **_GPIOCLRALL** ()

Clear GPIO pins.

Any pin that is assigned the value 1 is cleared. Pins that are given the value 0 are left as they are.

1.34 **_GPIOSETDIR** (UWORD pin, UWORD dir)

UWORD pin := pin to set the direction, Valid pin numbers are: 1,2,4,5,6,7,8,9.

UWORD dir := 0 for output, 1 for input

Sets the direction of the GPIO pin.

1.35 **UWORD _GPIOGETPIN** (UWORD pin)

UWORD pin := pin to get, Valid pin numbers are: 1,2,4,5,6,7,8,9.

Returns the value of the GPIO pin. 1 for asserted, 0 for de-asserted.

1.36 **UWORD _GPIOGETALL** ()

Returns the **value of the** all the GPIO pin.

1.37 **SLONG _SOCKBIND** (SLONG sockid , BYREF STRING addrPort)

SLONG sockid := Socket ID to perform the bind operation on.

BYREF STRING addrPort := Address and port to bind to.

Format: "ipaddr:port", ipaddr could be empty to denote INADDR_ANY.

Examples: "10.0.0.1:100" bind to address 10.0.0.1 and port 100

":100" bind to any address and port 100

This is a builtin function.

Binds a socket.

Returns 0 on success, non-zero on failure.

1.38 **SLONG _SOCKCONNECT** (SLONG sockid , BYREF STRING addrPort)

SLONG sockid := Socket ID.

BYREF STRING addrPort :=

Remote Address and port to connect to.

Format: "ipaddr:port"

Examples: "10.0.0.1:100" bind to address 10.0.0.1 and port 100

This is a builtin function.

Connects a socket.

Returns 0 on success, non-zero on failure.

1.39 **SLONG _SOCKLISTEN** (SLONG sockid)

SLONG sockid := Socket ID.

This is a builtin function.

Perform listen operation on socket.

Returns 0 on success, non-zero on failure.

1.40 **SLONG _SOCKACCEPT** (SLONG sockid , BYREF SLONG newsock) SLONG sockid

:= Socket ID.

BYREF SLONG newsock := Socket ID of the newly created socket.

This is a builtin function.

Perform accept operatin on the socket.

Returns 0 on success, non-zero on failure.

1.41 SLONG _SOCKCLOSE (SLONG sockid)
SLONG sockid := Socket ID.

This is a builtin function.

Close the socket.

Returns 0 on success, non-zero on failure.

1.42 SLONG _SOCKSENDALL (SLONG sockid , BYREF STRING data , ULONG flags)
SLONG sockid := Socket ID.
BYREF STRING data := The data to send.
ULONG flags := These flags are passed to the TCP/IP sendmsg() call as they are.

This is a builtin function.

Send data on a connected socket. It will return only when all of the data is written or an error occurs.

Returns 0 on success, non-zero on failure.

1.43 SLONG _SOCKSENDTO (SLONG sockid , BYREF STRING data , ULONG flags, BYREF STRING to)
SLONG sockid := Socket ID.
BYREF STRING data := The data to send.
ULONG flags := These flags are passed to the TCP/IP sendmsg() call as they are.
Set to 0, if you don't need any special behaviour.
BYREF STRING to := where to send the data to.
Format: "host:port".

This is a builtin function.

Send data on a connected socket. It will return only when all of the data is written or an error occurs.

Returns 0 on success, non-zero on failure.

1.44 SLONG _SOCKRECV (SLONG sockid , BYREF STRING data , ULONG flags, ULONG maxbytes)
SLONG sockid := Socket ID.
ULONG flags := These flags are passed to the TCP/IP recvmsg() call as they are.
Set to 0, if you don't need any special behaviour.
ULONG maxbytes := This function needs to allocate an internal buffer to queue the data into.

This should be set to biggest message expected. In TCP, specifying a small buflen will not result in loss of data. In UDP, if the incoming message is too big to fit into the internal buffer the message will be truncated and data will be lost.

This is a builtin function.

Recv data data from a connected socket. Will block waiting for data.
If an empty string is returned, then the socket has ben closed (EOF).
Returns 0 on success, non-zero on failure.

1.45 SLONG _SOCKRECVFROM (SLONG sockid, BYREF STRING data, ULONG flags, ULONG maxbytes, BYREF STRING from)
SLONG sockid := Socket ID.

BYREF STRING data := Place an empty string here.

On successful return, this string will contain the data read from the socket.

ULONG flags := These flags are passed to the TCP/IP `recvmsg()` call as they are.

Set to 0, if you don't need any special behaviour.

ULONG maxbytes := This function needs to allocate an internal buffer to queue the data into.

This should be set to biggest message expected. In TCP, specifying a small `buflen` will not result in loss of data. In UDP, if the incoming message is too big to fit into the internal buffer the message will be truncated and data will be lost.

BYREF STRING from := On successful return this string contains the IP address and port of sender in the form: `<ipaddr>:<port>`.

This is a builtin function.

Recv data from a connected socket. Will block waiting for data.

If an empty string is returned, then the socket has been closed (EOF).

Returns 0 on success, non-zero on failure.

1.46 SLONG _SOCKTCPSERVER (BYREF SLONG sockid , BYREF STRING addrPort)

BYREF SLONG sockid := Socket ID.

BYREF STRING addrPort := Address and port to bind to.

Format: "ipaddr:port", ipaddr could be empty to denote `INADDR_ANY`.

Examples: "10.0.0.1:100" bind to address 10.0.0.1 and port 100 ":100" bind to any address and port 100

This is a builtin function.

Creates a TCP server and blocks waiting for a client connection.

This is a convenience function. it doesn't do anything that cannot be done already.

It does the following: `socket(). bind(), listen() accept()`.

1.47 SLONG _SOCKTCPCLIENT (BYREF SLONG sockid , BYREF STRING addrPort)

BYREF SLONG sockid := Socket ID.

BYREF STRING addrPort := Remote Address and port to connect to.

Format: "ipaddr:port"

Examples: "10.0.0.1:100" bind to address 10.0.0.1 and port 100

This is a builtin function.

Creates a TCP socket and connects to a remote server.

This is a convenience function. it doesn't do anything that cannot be done already. It does the following: `socket(). connect()`. Returns 0 on success, non-zero on failure.

1.48 SLONG _SOCKGETNAME (SLONG sockid , BYREF STRING sockname)

SLONG sockid := Socket ID to perform the bind operation on.

BYREF STRING addrPort := Address and port that this socket is bound to.

Format: "ipaddr:port"

This is a builtin function.

Get the name of a socket.

Returns 0 on success, non-zero on failure.

1.49 SLONG _SOCKNONBLOCK (SLONG sockid)

SLONG sockid := Socket ID to put in non-blocking mode.

This is a builtin function.

Put the socket in non-blocking mode.

Returns 0 on success, non-zero on failure.

1.50 SLONG _SOCKTCPCREATE (BYREF SLONG sockid)
BYREF SLONG sockid := Reference to a socket ID.
On success, 'sockid' will contain the ID of the newly created socket.

This is a builtin function.
Creates a TCP socket.
Returns 0 on success, non-zero on failure.

1.51 SLONG _SOCKUDPCREATE (BYREF SLONG sockid, UWORD broadcast)
BYREF SLONG sockid := Reference to a socket ID. On success, 'sockid' will contain the ID of the newly created socket.
UWORD broadcast := Set to non-zero to request a socket that is able to send and receive broadcast packets.

This is a builtin function.
Creates a UDP socket. Returns 0 on success, non-zero on failure.

1.52 SLONG _MYGWADDRGET (BYREF STRING gwaddr)
BYREF STRING gwaddr := The IP address of the default gateway is written into this string.

This is a builtin function.
Gets the default gateway IP address associated with the TCP/IP stack.
Returns 0 on success, non-zero on failure.

1.53 STRING _GETMACADDR ()
Returns the MAC address of the wireless interface.
The MAC address is returned in binary form, not in string form.

1.54 STRING _GETMYIPADDR\$ ()

This is a builtin function.
Returns the IP address associated with the wireless LAN interface.
If there are multiple IP addresses the first one found is returned.
Returns an empty string, if no IP address exists

1.55 STRING _GETMYBRDADDR\$ ()

This is a builtin function.
Returns the broadcast IP address associated with the wireless LAN interface.
If there are multiple addresses the first one found is returned.
Returns an empty string if no broadcast address is found.

1.56 STRING _GETMYGWADDR\$ ()

This is a builtin function.
Returns the default gateway of the TCP/IP stack.
Returns an empty string if no gateway entry is found.

1.57 SLONG _MYIPADDRGET (BYREF STRING ipaddr)
BYREF STRING ipaddr := The IP address is written into this string.

This is a builtin function.
Get the IP address associated with the wireless LAN interface. If there are multiple IP addresses the first one found is returned.
Returns 0 on success, non-zero on failure.

- 1.58 SLONG _MYBCASTADDRGET** (BYREF STRING brdaddr)
BYREF STRING ipaddr := The broadcast IP address is written into this string.
- This is a builtin function.
Get the broadcast IP address associated with the wireless LAN interface. If there are multiple addresses the first one found is returned.
Returns 0 on success, non-zero on failure.
- 1.59 _DHCPSETSTATE** (UWORD flag)
UWORD flag := set to one to enable DHCP on the wireless interface, 0 to disable
- 1.60 _MYIPADDRSET** (STRING addr, STRING netmask)
Adds the IP interface specified by addr and netmask to the wireless interface.
- 1.61 _MYIPADDRDEL** ()
Deletes the IP addresses from the wireless interface.
If no IP address exists, it does nothing.
- 1.62 _DNSCONFGET** (BYREF STRING domain, BYREF STRING dns1, BYREF STRING dns2)
domain := domain name e.g. "ezurio.com" dns1 := IP address of first DNS server in dotted format e.g. "10.0.0.1" dns2 := IP address of second DNS server in dotted format e.g. "10.0.0.1" Get the DNS configuration settings.
- 1.63 _DNSCONFSET** (STRING domain, STRING dns1, STRING dns2)
domain := domain name e.g. "ezurio.com" dns1 := IP address of first DNS server in dotted format e.g. "10.0.0.1" dns2 := IP address of second DNS server in dotted format e.g. "10.0.0.1" Set the DNS configuration settings.
- 1.64 ULONG FOPEN** (STRING BYVAL filename, UWORD mode)
STRING BYVAL filename := Full path of file to open
UWORD mode := Type of access permitted as follows:
0 Opens for reading. If the file does not exist or cannot be found, the fopen call fails.
Other Values Will fail
- This is a builtin function.
Returns a handle to the file. Will be 0 if file could not be opened
- 1.65 UWORD FCLOSE** (ULONG fileHandle)
ULONG fileHandle:= Value that was returned by FOPEN
- This is a builtin function.
Returns 0 if the file was closed successfully. Other values signify error
- 1.66 UWORD FREAD** (ULONG fileHandle, STRING BYREF dataStr, UWORD nMaxRead)
ULONG fileHandle:= Value that was returned by FOPEN
STRING BYREF dataStr:= String variable to read the data from the file
UWORD nMaxRead:= Bytes to read from the file
- This is a builtin function.
Returns the number of bytes read from the file.
Will be 0 if the end of file reached. or if the file handle is invalid
- 1.67 _FLASHDEL** ()

EZURiO

This is a builtin function.

Deletes the contents of the FLASH file system.

The flash file system is used to store the UW scripts and other data files. After this you must reboot

1.68 ONEVENT symbolic names

EV0	=	Event Number 0 to 8 are reserved for future use.
EVSTDIN	=	New data has arrived in STDIN
EVRXBRKON	=	Uart has detected a BREAK
EVRXBRKOFF	=	Uart has detected a BREAK release
EVWEB	=	Web Server needs servicing
EVSEARCH	=	A search for access points has completed
EVSOCKETCLOSE	=	A pre-existing socket has closed
EVLINK	=	Network Link layer status changed

2.0 TCP/IP Error Codes

#define IP_ERRNO_EPERM	1	
#define IP_ERRNO_ENOENT	2	
#define IP_ERRNO_ESRCH	3	
#define IP_ERRNO_EINTR	4	
#define IP_ERRNO_EIO	5	
#define IP_ERRNO_ENXIO	6	
#define IP_ERRNO_E2BIG	7	
#define IP_ERRNO_ENOEXEC	8	
#define IP_ERRNO_EBADF	9	
#define IP_ERRNO_ECHILD	10	
#define IP_ERRNO_EAGAIN	11	
#define IP_ERRNO_EWOULDBLOCK	IP_ERRNO_EAGAIN	
#define IP_ERRNO_ENOMEM	12	
#define IP_ERRNO_EACCES	13	
#define IP_ERRNO_EFAULT	14	
#define IP_ERRNO_ENOTBLK	15	
#define IP_ERRNO_EBUSY	16	
#define IP_ERRNO_EEXIST	17	
#define IP_ERRNO_EXDEV	18	
#define IP_ERRNO_ENODEV	19	
#define IP_ERRNO_ENOTDIR	20	
#define IP_ERRNO_EISDIR	21	
#define IP_ERRNO_EINVAL	22	
#define IP_ERRNO_ENFILE	23	
#define IP_ERRNO_EMFILE	24	
#define IP_ERRNO_ENOTTY	25	
#define IP_ERRNO_ETXTBSY	26	
#define IP_ERRNO_EFBIG	27	
#define IP_ERRNO_ENOSPC	28	
#define IP_ERRNO_ESPIPE	29	
#define IP_ERRNO_EROFS	30	
#define IP_ERRNO_EMLINK	31	
#define IP_ERRNO_EPIPE	32	
#define IP_ERRNO_EDOM	33	
#define IP_ERRNO_ERANGE	34	
#define IP_ERRNO_ELOCAL	35	
#define IP_ERRNO_EINPROGRESS	36	<i>/* Operation now in progress */</i>
#define IP_ERRNO_EALREADY	37	<i>/* Operation already in progress */</i>
#define IP_ERRNO_ENOTSOCK	38	<i>/* Socket operation on non-socket */</i>
#define IP_ERRNO_EDESTADDRREQ	39	<i>/* Destination address required */</i>
#define IP_ERRNO EMSGSIZE	40	<i>/* Message too long */</i>
#define IP_ERRNO_EPROTOTYPE	41	<i>/* Protocol wrong type for socket */</i>
#define IP_ERRNO_ENOPROTOOPT	42	<i>/* Protocol not available */</i>
#define IP_ERRNO_EPROTONOSUPPORT	43	<i>/* Protocol not supported */</i>
#define IP_ERRNO_ESOCKTNOSUPPORT	44	<i>/* Socket type not supported */</i>
#define IP_ERRNO_EAFNOSUPPORT	47	<i>/* Address family not supported */</i>
#define IP_ERRNO_EADDRINUSE	48	<i>/* Address already in use */</i>
#define IP_ERRNO_EADDRNOTAVAIL	49	<i>/* Can't assign requested address */</i>
#define IP_ERRNO_ENETDOWN	50	<i>/* Network is down */</i>
#define IP_ERRNO_ENETUNREACH	51	<i>/* Network is unreachable */</i>
#define IP_ERRNO_ENETRESET	52	<i>/* Network dropped conn. on reset */</i>
#define IP_ERRNO_ECONNABORTED	53	<i>/* Software caused connection abort */</i>
#define IP_ERRNO_ECONNRESET	54	<i>/* Connection reset by peer */</i>
#define IP_ERRNO_ENOBUFS	55	<i>/* No buffer space available */</i>
#define IP_ERRNO_EISCONN	56	<i>/* Socket is already connected */</i>
#define IP_ERRNO_ENOTCONN	57	<i>/* Socket is not connected */</i>
#define IP_ERRNO_ESHUTDOWN	58	<i>/* Can't send after socket shutdown */</i>
#define IP_ERRNO_ETOOMANYREFS	59	<i>/* Too many references: can't splice */</i>

```
#define IP_ERRNO_ETIMEDOUT          60    /* Operation timed out */
#define IP_ERRNO_ECONNREFUSED      61    /* Connection refused */
#define IP_ERRNO_EHOSTDOWN        64    /* Host is down */
#define IP_ERRNO_EHOSTUNREACH     65    /* No route to host */
#define IP_ERRNO_EPROCLIM         67    /* Too many processes */
#define IP_ERRNO_EUSERS           68    /* Too many users */
#define IP_ERRNO_EDQUOT           69    /* Disc quota exceeded */
#define IP_ERRNO_EBADR           70    /* Invalid request descriptor */
#define IP_ERRNO_EFAIL           85
#define IP_ERRNO_ENODATA         86    /* No data available */
#define IP_ERRNO_EILSEQ          88    /* Illegal byte sequence */

#define IP_ERRNO_EDEADLK         45
#define IP_ERRNO_ENOLCK         46
#define IP_ERRNO_ENAMETOOLONG   78
#define IP_ERRNO_ENOSYS         89
#define IP_ERRNO_ENOTEMPTY     158
#define IP_ERRNO_EOPNOTSUPP    145
#define IP_ERRNO_EPFNOSUPPORT   146
#define IP_ERRNO_ERRMAX        159
```

Appendix 1 - SAMPLE SCRIPTS

1. Simple script to automatically attach to an access point

```
string stSSID
uword uwRes
slong slRes

stSSID = "3Com"

_wlankey("167A80D55FC54F1CE5C19D76C1")
_wlansecurity(1)
uwRes = _wlanattach(stSSID)

if (uwRes != 0) then
    print "Failed to attach to network """"; stSSID; """"\n"
    stSSID = "dimitris"
    uwRes = _wlanattach(stSSID)
    if (uwRes != 0) then
        print "Failed to attach to network\09""""; stSSID; """"\n"
    else
        print "Attached to network\09""""; stSSID; """"\n"
    endif
else
    print "Attached to network """"; stSSID; """"\n"
endif
```



```
    // keep echoing data until the client closes the connection
    finish = 0
    while finish == 0
        sRes = _sockrecv(sIDataSock, stRecvBuf, 0, 100)
        if sRes != 0 then
            print "recv failed: sRes="; sRes; "\n"
            print "closing connection\n"
            finish = 1
        endif
        if strlen(stRecvBuf) == 0 then
            print "client closed connection\n"
            finish = 1
        endif
        sRes = _socksendall(sIDataSock, stRecvBuf, 0)
        if sRes != 0 then
            print "send failed: sRes="; sRes; "\n"
            print "closing connection\n"
            finish = 1
        endif
    endwhile

    print "after dountil loop\n"

    // close the data socket
    print "before close\n"
    sRes = _sockclose(sIDataSock)
    print "after close\n"
    if sRes != 0 then
        print "close of data socket failed: sRes="; sRes; "\n"
        sRes = _sockclose(sILstnSock)
        exitsub
    endif

endwhile

// close the listening socket
sRes = _sockclose(sILstnSock)
if sRes != 0 then
    print "close of listening socket failed: sRes="; sRes; "\n"
    exitsub
endif

endsub

// Run the server
EchoServer()
```

3. Simple script to report AP RSSI

```
'// Paramaters: none
'//
string stRecvBuf
uword result
uword value
string stMyip
uword avgsnr
uword instsnr
uword avgrssi
uword instrssi

_wlansearchdump()
result = _wlanattach("3Com")

if (result != 0) then
    print "Failed to attach to network 3Com\n"
endif
while 1
    _rssiaget(avgsnr, instsnr, avgrssi, instrssi)
    print "Average SNR = "; avgsnr; "\r"
endwhile
```

4. Simple script to monitor all GPIOs

```
///  
// Paramaters: none  
///  
uword pio_state  
  
_gpiosetdir(1, 1)  
_gpiosetdir(2, 1)  
_gpiosetdir(4, 1)  
_gpiosetdir(5, 1)  
_gpiosetdir(6, 1)  
_gpiosetdir(7, 1)  
_gpiosetdir(8, 1)  
_gpiosetdir(9, 1)  
while 1  
    pio_state = _gpiogetall()  
    print "PIO state is "; pio_state; "\n"  
endwhile
```

5. Simple script to toggle through all GPIOs

```
///  
// Paramaters: none  
///  
_gpiosetdir(1, 0)  
_gpiosetdir(2, 0)  
_gpiosetdir(4, 0)  
_gpiosetdir(5, 0)  
_gpiosetdir(6, 0)  
_gpiosetdir(7, 0)  
_gpiosetdir(8, 0)  
_gpiosetdir(9, 0)  
while 1  
    _gpiosetpin(1)  
    sleep(50)  
    _gpioclrpin(1)  
    _gpiosetpin(2)  
    sleep(50)  
    _gpioclrpin(2)  
    _gpiosetpin(4)  
    sleep(50)  
    _gpioclrpin(4)  
    _gpiosetpin(5)  
    sleep(50)  
    _gpioclrpin(5)  
    _gpiosetpin(6)  
    sleep(50)  
    _gpioclrpin(6)  
    _gpiosetpin(7)  
    sleep(50)  
    _gpioclrpin(7)  
    _gpiosetpin(8)  
    sleep(50)  
    _gpioclrpin(8)  
    _gpiosetpin(9)  
    sleep(50)  
    _gpioclrpin(9)  
endwhile
```


6. Web Event Handling

```
// attach to the network
function sword mStrCompare(string byref s1, string byval s2)
endfunc strcmp(s1,s2)

subroutine helloWorld(string byval addthisstring)
    string thisString
    thisString = "Hello World "
    print thisString; addthisstring
endsub

function uword webhandler()

string  scriptName
string  extension
string  compareScript
slong   slRes
uword   status
ulong   ulFile
string  stFile
string  stData
uword   evCause
string  postName

    // get the name of the script/file for this event occurring
    slRes = _webFNameGet(scriptName)

    // get the extension type
    slRes = _webFExtGet(extension)

    // get the cause of the web event
    slRes = _webEvGet(evCause)

    // check the cause 1 = UwScript or File, 2 = a POST was made to the server from the
    browser
    if evCause == 1 then
        // it was a file or UwScript call
        print "Event caused by UwScript or file get.\n"
        // check the extension
        if mStrCompare(extension, "uws") == 0 then
            // it was a script so check the name of the script and call the script function
            if mStrCompare(scriptName, "helloWorld") == 0 then
                helloWorld("We've created a web event!!\n")
                // we want to print something to the web browser so first
                _webEchoStart()
                webecho "We've created a web event!!\n"
                // we have finished so end the web event
                slRes = _WebEvEnd()
            endif
        elseif mStrCompare(scriptName, "stopScript") == 0 then
            print "The script is now stopping at the request of the web server\n"
            // we want to print something to the web browser so first
            _webEchoStart()
            webecho "Stop script activated.\n"
            webecho "When you next visit this page you will see different
contents.<br>\n"
```

```
again.\n"
    webecho "Unless you re-start the event handler by running this script
again.\n"
    '// we have finished so end the web event
    slRes = _WebEvEnd()
    exitfunc 0
else
    '// we don't have a matching script for that tag so just print HELLO
    helloWorld("ERROR No reason for web event\n")
    '// we have finished so end the web event
    slRes = _WebEvEnd()
    exitfunc 1
endif
elseif mStrCompare(extension, "html") == 0 then
    print "\nBefore File Open\n"
    stFile=scriptName + "." + extension
    print "opening " ; stFile ; "\n"
    '// open the file
    ulFile=fopen(stFile,0)
    '// we want to send data to the web browser so must first call
    _webechostart()
    do
        status=fread(ulFile,stData,500)
        webecho stData
    until status == 0

    '// close the file
    status=fclose(ulFile)
    print "\nAfter File Close\n"
    '// we have finished so end the web event
    slRes = _WebEvEnd()
    exitfunc 1
endif
elseif evCause == 2 then
    '// it was a POST, so fetch the POST data
    slRes = _webPostGet(postName, scriptName)
    '// We are just going to print it to the UwTerminal display for now
    print "Received post "; postName; " and " ; scriptName; " posted from the
browser\n"

    '// to create this dynamic html we will get the main part of the page from the file
system
    ulFile=fopen("post_page_one.html",0)
    if (ulfile) then
        _webechostart()
        do
            status=fread(ulFile,stData,500)
            webecho stData
        until status == 0

        '// close the file
        status=fclose(ulFile)

        '// now send the dynamic content
        webecho "<div id=""embed"">"
        webecho "<p>Here is the post name and text dynamically generated into this
web page"
```

EZURiO

```
        webecho "<br><h2>Post Name - ", postName ; "<br>Post Text - ",
scriptName; "</h2></p></div></body></html>"
    endif

    '// we have finished so end the web event
    slRes = _WebEvEnd()
    exitfunc 1
else
    print "Error - unkown web event detected\n"
    '// we have finished so end the web event
    slRes = _WebEvEnd()
    exitfunc 1
endif

    slRes = _WebEvEnd()
endfunc 1

'// send an initial message
helloWorld("I'm going to start the web event handler\n")

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

7. HTTP GET

```
subroutine httpget(string stAddr, string stPath)
  string stRcvdMsg
  string stHttpReq
  string stDest
  slong slRes
  slong slSock
  ulong i
  string stMyip

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

  stDest = stAddr + ":80"
  slRes = _socktcpclient(slSock, stDest)
  if slRes != 0 then
    print "Failed to create simple client: slRes="; slRes; "\n"
    exitsub
  endif

  stHttpReq = "GET " + stPath + " HTTP/1.0\n\r\n"
  slRes = _socksendall(slSock, stHttpReq, 0)
  if slRes != 0 then
    print "Failed to send: slRes="; slRes; "\n"
    exitsub
  endif

  stRcvdMsg = "x"
  while strlen(stRcvdMsg) != 0

    slRes = _sockrecv(slSock, stRcvdMsg, 0, 100)
    if slRes != 0 then
      print "Failed to recv: slRes="; slRes; "\n"
      exitsub
    endif
    print stRcvdMsg

  endwhile

  slRes = _sockclose(slSock)
  if slRes != 0 then
    print "Failed to close socket: slRes="; slRes; "\n"
    exitsub
  endif

endsub

string stAddr
string stPath

stAddr = "10.0.0.1"
stPath = "/"

httpget(stAddr, stPath)
```

