

Native Wifi Functions v3.2

for XP SP3
~~~ **By MattyD** ~~~

| <i>Session Management</i>                    |                                                                                                 |    |
|----------------------------------------------|-------------------------------------------------------------------------------------------------|----|
| <code>_Wlan_StartSession()</code>            | Calls <code>OpenHandle</code> , <code>EnumInterfaces</code> and <code>SetGlobalConstants</code> | 16 |
| <code>_Wlan_EndSession()</code>              | Calls <code>CloseHandle</code> and closes <code>WlanAPI.dll</code>                              | 16 |
| <code>_Wlan_OpenHandle()</code>              | Returns a client handle for following functions                                                 | 16 |
| <code>_Wlan_CloseHandle()</code>             | Closes a client handle                                                                          | 17 |
| <code>_Wlan_EnumInterfaces()</code>          | Enumerates and gathers interface information (incl. <code>\$pGUID</code> )                      | 17 |
| <code>_Wlan_SetGlobalConstants()</code>      | Sets default <code>\$pGUID</code> and <code>\$hClientHandle</code> values                       | 17 |
| <i>Connection Management</i>                 |                                                                                                 |    |
| <code>_Wlan_Scan()</code>                    | Initiates a scan for wireless networks                                                          | 17 |
| <code>_Wlan_GetAvailableNetworkList()</code> | Returns information on networks in range                                                        | 18 |
| <code>_Wlan_Connect()</code>                 | Connects to a network if a corresponding profile exists                                         | 19 |
| <code>_Wlan_Disconnect()</code>              | Disconnects from a network                                                                      | 19 |
| <code>_Wlan_ConnectWait()</code>             | As <code>_Wlan_Connect()</code> but waits for connection to complete                            | 19 |
| <code>_Wlan_DisconnectWait()</code>          | As <code>_Wlan_Disconnect()</code> but waits for connection to cease                            | 19 |
| <code>_Wlan_WaitForDisconnect()</code>       | Idles until disconnected                                                                        | 20 |
| <i>Profile Management</i>                    |                                                                                                 |    |
| <code>_Wlan_GetProfile()</code>              | Returns information about an existing profile                                                   | 20 |
| <code>_Wlan_SetProfile()</code>              | Creates a new profile                                                                           | 20 |
| <code>_Wlan_GetProfileXML()</code>           | Returns information about an existing profile in XML format                                     | 20 |
| <code>_Wlan_SetProfileXML()</code>           | Creates a new profile from an XML format                                                        | 20 |
| <code>_Wlan_DeleteProfile()</code>           | Purges an existing profile                                                                      | 21 |
| <code>_Wlan_GetProfileList()</code>          | Returns a list of profiles in order of priority                                                 | 21 |
| <code>_Wlan_SetProfileList()</code>          | Sets existing profiles in order of priority                                                     | 21 |
| <code>_Wlan_SetProfilePosition()</code>      | Changes the priority of a nominated profile                                                     | 21 |
| <i>Interface Management</i>                  |                                                                                                 |    |
| <code>_Wlan_QueryInterface()</code>          | Returns various interface settings or connection information                                    | 21 |
| <code>_Wlan_SetInterface()</code>            | Sets various interface parameters                                                               | 23 |
| <i>Toolbox</i>                               |                                                                                                 |    |
| <code>_Wlan_GenerateXMLProfile()</code>      | Generates an XML profile from an array format                                                   | 23 |
| <code>_Wlan_StringToPguid()</code>           | Returns a <code>\$pGUID</code> value from a GUID string                                         | 23 |
| <code>_Wlan_PguidToString()</code>           | Returns a GUID string from a <code>\$pGUID</code> value                                         | 23 |

|          |
|----------|
| New      |
| Modified |
| Obsolete |

## Table of Contents

|                                       |    |
|---------------------------------------|----|
| Included Files.....                   | 3  |
| Requirements.....                     | 3  |
| New Stuff.....                        | 3  |
| Installer Options .....               | 4  |
| Error Management .....                | 5  |
| Behavioral Remarks.....               | 6  |
| Profile Array Format.....             | 7  |
| Profile Structures.....               | 8  |
| Profile Array/GUI Relationships.....  | 10 |
| Example Profiles.....                 | 13 |
| Functions.....                        | 16 |
| _Wlan_StartSession() .....            | 16 |
| _Wlan_EndSession().....               | 16 |
| _Wlan_OpenHandle().....               | 16 |
| _Wlan_CloseHandle() .....             | 17 |
| _Wlan_EnumInterfaces() .....          | 17 |
| _Wlan_SetGlobalConstants().....       | 17 |
| _Wlan_Scan().....                     | 17 |
| _Wlan_GetAvailableNetworkList() ..... | 18 |
| _Wlan_Connect().....                  | 19 |
| _Wlan_Disconnect() .....              | 19 |
| _Wlan_ConnectWait() .....             | 19 |
| _Wlan_DisconnectWait() .....          | 19 |
| _Wlan_WaitForDisconnect() .....       | 20 |
| _Wlan_GetProfile() .....              | 20 |
| _Wlan_SetProfile() .....              | 20 |
| _Wlan_GetProfileXML() .....           | 20 |
| _Wlan_SetProfileXML() .....           | 20 |
| _Wlan_DeleteProfile() .....           | 21 |
| _Wlan_GetProfileList() .....          | 21 |
| _Wlan_SetProfileList() .....          | 21 |
| _Wlan_SetProfilePosition() .....      | 21 |
| _Wlan_QueryInterface() .....          | 21 |
| _Wlan_SetInterface() .....            | 23 |
| _Wlan_GenerateXMLProfile().....       | 23 |
| _Wlan_StringTopGUID() .....           | 23 |
| _Wlan_pGUIDToString() .....           | 23 |

## Included Files

### Common

|                                   |                                           |
|-----------------------------------|-------------------------------------------|
| NativeWifi_Setup32.exe            | File installer                            |
| \4SciTE\keywords32.txt            | SciTE keywords source for installer SciTE |
| \Support\NativeWifi_Help32.pdf    | This help file                            |
| \Support\NativeWifi_InstOpt32.pdf | Installer help file                       |

### Version A

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| Wifi32a.au3                       | Autolt3 UDF                         |
| \4SciTE\Calltips32a.txt           | SciTE calltips source for installer |
| \Support\NativeWifi_Sample32a.au3 | Example script with tips            |

### Version B (ver. 2 syntax)

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| Wifi32b.au3                       | Autolt3 UDF                         |
| \4SciTE\Calltips32b.txt           | SciTE calltips source for installer |
| \Support\NativeWifi_Sample32b.au3 | Example script with tips            |

## Requirements

### OS

XP SP3 or SP2  
limited functionality on Vista / 7...

### Hotfixes

*XP SP2:*  
<http://support.microsoft.com/kb/918997>

*XP SP3:*  
<http://support.microsoft.com/kb/958071>

### Services

Windows Zero Configuration (wzcsvc)  
Network Connections (netman)

## New Stuff

### Added

Function \_Wlan\_WaitForDisconnect()  
Function \_Wlan\_SetProfileXML()  
Function \_Wlan\_GetProfileXML()  
Function \_Wlan\_GenerateXMLProfile()  
EAP functionality  
Installer  
.pdf help file  
Extended Vista\7 functionality (unsupported)

### Changed

Code cleanup (all functions)  
Function \_Wlan\_SetProfile()  
Function \_Wlan\_GetProfile()  
Function \_Wlan\_StringTopGUID()  
.au3 sample file

### Removed

.au3 help file

## Installer Options

### Use Version x

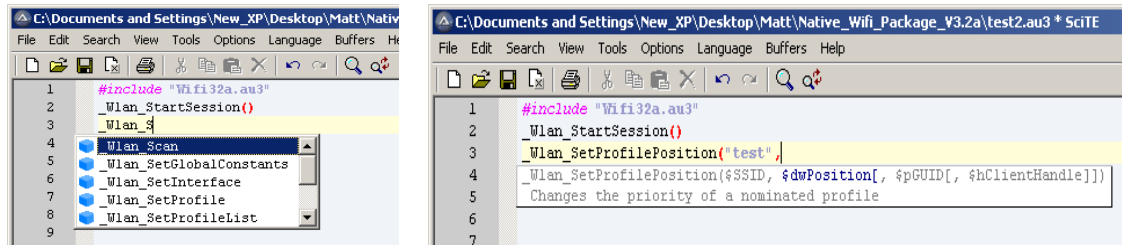
Version b uses the old V2 syntax for backward compatibility. It is highly recommended Version a is used to avoid the constant typing redundant parameters.

### Add to Include Folder

This option copies the UDF to Autolt's Include folder to avoid wasting time finding it again further down the track (allows you to use the `#include <wifi32a.au3>` code).

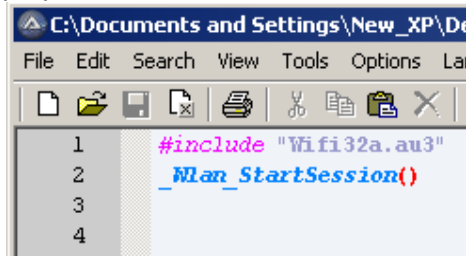
### Use Call Tips

This option adds call information to the file `au3.user.calltips.api` located in SciTE's api folder. This both allows function names to be found quickly and provides a quick reference to a function's syntax.



### Use UDF Highlighting

This option adds the UDF's function names to the file `au3.userudfs.properties` located in SciTE's properties folder. This causes the corresponding function names to be highlighted in SciTE.



### Open Help File

Opens this PDF file

### Open Sample

Opens sample code

## Error Management

**@error = 0**

Success!

**@error = 1**

Dll Call Error

The function returns a reason for the error

**@Extended** is set to the Dll error code

To interpret the @extended value go to:

<http://msdn.microsoft.com/en-us/library/ms681381.aspx>

**@error = 2**

Number of interfaces/available networks/profiles/profile names = 0

**@error = 3**

Invalid Parameter/Profile (Dll was not called)

**@error = 4**

Unable to use the Dll

**@error = 5**

Function timed out

## Behavioral Remarks

### Default Client Handle and GUID Pointer Values

A client handle (referred to as \$hClientHandle) and GUID pointer (ref. as \$pGUID) are required for the majority of functions in this UDF. These parameters can be assigned default values by calling `_Wlan_StartSession()` or `_Wlan_SetGlobalDefaults()`.

Once the default values have been assigned:

- |               |                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------|
| Version a     | \$hClientHandle and \$pGUID parameters are optional for subsequent functions.                     |
| Both versions | -l or the Default keyword can be used as a substitute for \$hClientHandle and \$pGUID parameters. |

A \$hClientHandle value is obtained through calling `_Wlan_StartSession()` or `_Wlan_OpenHandle()`.

A \$pGUID value is obtained through calling `_Wlan_StartSession()`, `_Wlan_EnumInterfaces()` or `_Wlan_StringToGUID()`.

### Close Your Handle!

Any open client handle(s) should be closed in the Autolt exit function (default `OnAutoltExit()`) especially when using `_Wlan_ConnectWait()`, `_Wlan_ConnectWait()` or `_Wlan_WaitForDisconnect()` in the script. Failure to do so may stop the Autolt process closing when working in the debugger. In this case it is necessary to manually kill the process Autolt3.exe.

It is preferable to use `_Wlan_EndSession()` to `_Wlan_CloseHandle()` in the exit function as it also closes the dll.

### Ad Hoc Profiles

Ad hoc profiles names are generated from the SSID of the network followed by "-ad hoc". This suffix should be generally be added on the \$SSID parameter for profile related functions (including `_Wlan_Connect()`). This should not be added on for `_Wlan_SetProfile()`.

As a rule, ad hoc profiles always appear after the infrastructure profiles in the profile list. If an attempt is made to change this with `SetProfilePosition` or `SetProfileList`, the functions will succeed but WZC will push them back to fit this criterion.

### Other Remarks

`_Wlan_EnumInterfaces()` and `_Wlan_QueryInterface()` will show "Connected" once a connection is made with the host - before acquiring an IP Address. `_Wlan_ConnectWait()` will wait until IP is acquired.

Auto Config (AKA the "use Windows to configure my wireless network settings" checkbox) needs to be enabled for many functions to work. This can be done through `_Wlan_SetInterface()`.

`_Wlan_GetProfile()` and `_Wlan_GetProfileXML()` will always return EAP configuration data as a blob (binary large object).

A function returning "The system cannot find the file specified." (@error = 1 and @extended = 2) is caused by an invalid \$pGUID value.

## Profile Array Format

Profiles in array format can be either have one or two subscripts and are no longer bound to a fixed number of elements. EAP structures should be added to the first dimension of a profile array as needed (see examples on page 13). Generally the order in which these structures are placed is not important; however there are a couple of exceptions.

1. The Base profile structure must always be first.
2. If PEAP-TLS is used, the first Server Validation structure refers to the PEAP configuration while the second refers to the TLS.

The Blob element is a binary representation of EAP configuration of data. If present, the blob element will override any other settings present in the profile.

The Default Blob is the configuration Windows uses by default when the “The key is provided for me automatically” checkbox is checked.

The key type should be set to Network Key if either:

1. Key material value is in a hex format
2. Authentication is set to Shared Key
3. Encryption is set to WEP

The key type should be set to Pass Phrase if the authentication method is WPA-PSK or WPA2-PSK and the key material in ASCII format.

`_Wlan_GetProfile()` will always return a 1 dimensional, 11 element array.

## Profile Structures

| <i>Base Profile</i> |                 |                                                          |                      |
|---------------------|-----------------|----------------------------------------------------------|----------------------|
| <i>Index</i>        | <i>Field</i>    | <i>Possible Values</i>                                   | <i>Default Value</i> |
| [0]                 | SSID            |                                                          |                      |
| [1]                 | Network type    | Infrastructure<br>Ad Hoc                                 | Infrastructure       |
| [2]                 | Connection type | Automatic<br>Manual                                      | Automatic            |
| [3]                 | Authentication  | Open<br>Shared Key<br>WPA<br>WPA-PSK<br>WPA2<br>WPA2-PSK | Open                 |
| [4]                 | Encryption      | Unencrypted<br>WEP<br>TKIP<br>AES                        | Unencrypted          |
| [5]                 | 802.1x status   | 802.1x Enabled<br>802.1x Disabled                        | 802.1x Disabled      |
| [6]                 | Key type        | No Key Material<br>Network Key<br>Pass Phrase            | No Key Material      |
| [7]                 | Key material    |                                                          | No Key Material      |
| [8]                 | Key index       | No Key Index<br>1<br>2<br>3<br>4                         | No Key Index         |
| [9]                 | EAP blob type   | No Blob<br>Default Blob<br>PEAP<br>TLS                   | No Blob              |
| [10]                | Blob            |                                                          | No Blob              |



| <i>Server Validation</i> |                                   |                                                    |
|--------------------------|-----------------------------------|----------------------------------------------------|
| <b>Index</b>             | <b>Field</b>                      | <b>Possible Values</b>                             |
| [0]                      | Identifier                        | Validate Certificate<br>Don't Validate Certificate |
| [1]                      | Prompt user for server validation | Prompt User<br>Don't Prompt User                   |
| [2]                      | Server names                      |                                                    |
| [3 - ...]                | Certificate thumbprints           |                                                    |

| <i>TLS</i>   |                                    |                                         |
|--------------|------------------------------------|-----------------------------------------|
| <b>Index</b> | <b>Field</b>                       | <b>Possible Values</b>                  |
| [0]          | Identifier                         | TLS                                     |
| [1]          | Certificate / smart card           | Certificate<br>Smart Card               |
| [2]          | Simple certificate selection       | Simple Selection<br>No Simple Selection |
| [3]          | Different user name for connection | Different User Name<br>Same User Name   |

| <i>PEAP</i>  |                                           |                                                      |
|--------------|-------------------------------------------|------------------------------------------------------|
| <b>Index</b> | <b>Field</b>                              | <b>Possible Values</b>                               |
| [0]          | Identifier                                | PEAP                                                 |
| [1]          | Fast reconnect                            | Fast Reconnect<br>No Fast Reconnect                  |
| [2]          | Quarantine checks                         | Quarantine Checks<br>No Quarantine Checks            |
| [3]          | Disconnect if no cryptobinding is present | Require Cryptobinding<br>Don't Require Cryptobinding |

| <i>MSCHAP</i> |                                |                                      |
|---------------|--------------------------------|--------------------------------------|
| <b>Index</b>  | <b>Field</b>                   | <b>Possible Values</b>               |
| [0]           | Identifier                     | MSCHAP                               |
| [1]           | Use Windows login and password | Use Win Logon<br>Don't Use Win Logon |

Profile Array/GUI Relationships

Always Checked

☐ Connect even if this network is not broadcasting

Base[8]  
Key Index

Base[9]  
Default Blob

Base[1]  
Network Type

Wireless network key

This network requires a key for the following:

Network Authentication: Open

Data encryption: WEP

Network key:

Confirm network key:

Key index (advanced): 1

☐ The key is provided for me automatically

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Base[0]  
SSID

Base[3]  
Authentication

Base[4]  
Encryption

Base[7]  
Key Material

Wireless network properties

Association Authentication Connection

Network name (SSID):

OK Cancel

Base[5]  
802.1x Status

Always Checked

Never Checked

Enable IEEE 802.1x authentication for this network

EAP type: Smart Card or other Certificate

Properties

☒ Authenticate as computer when computer information is available

☐ Authenticate as guest when user or computer information is unavailable

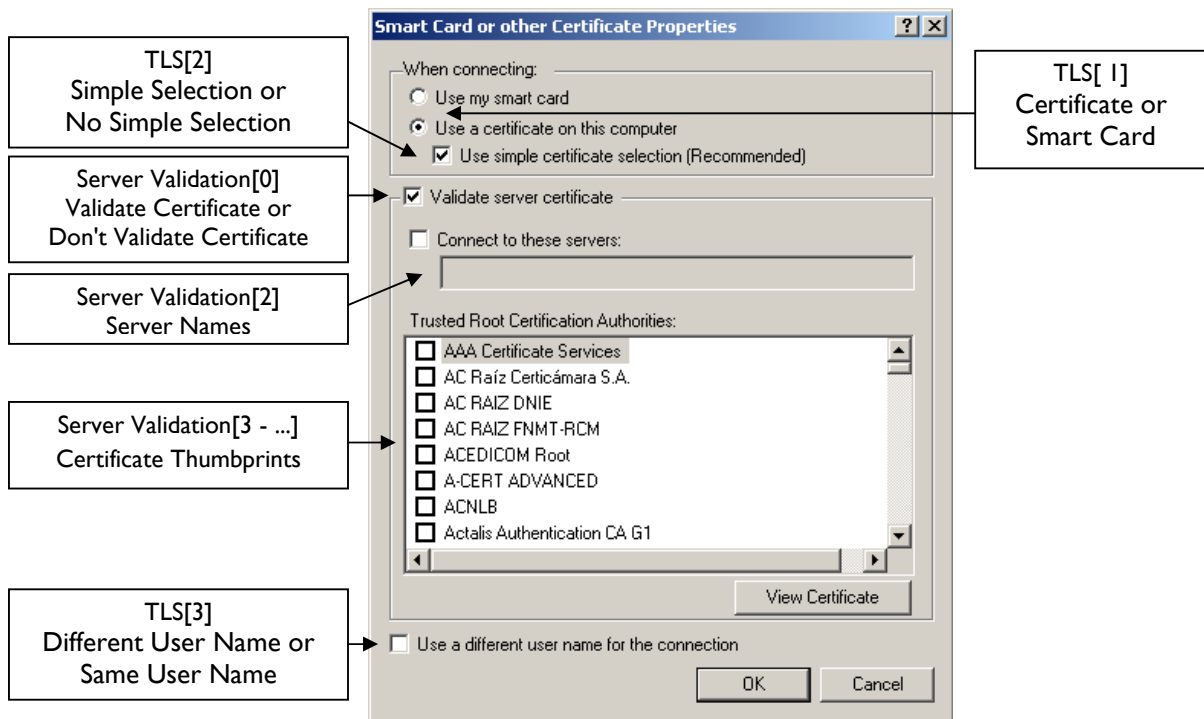
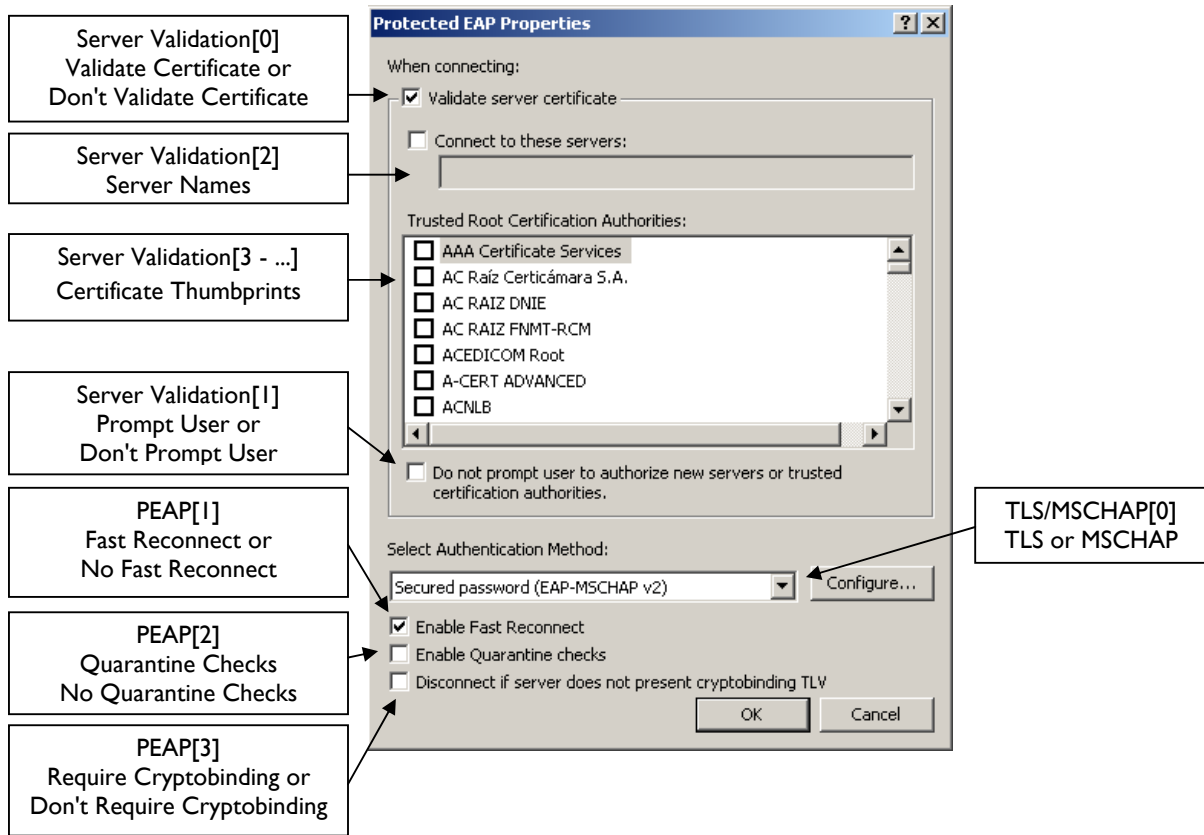
PEAP/TLS[0]  
TLS or PEAP

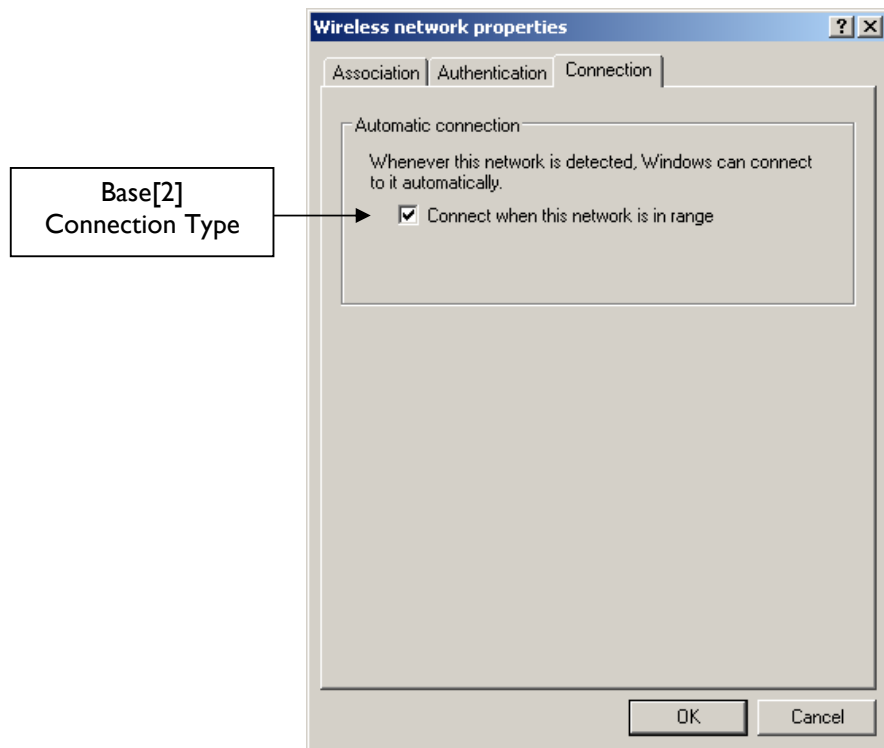
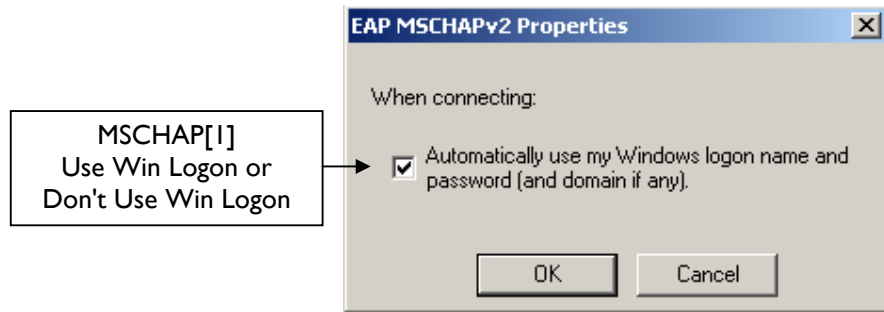
Wireless network properties

Association Authentication Connection

Select this option to provide authenticated network access for wireless Ethernet networks.

OK Cancel





## Example Profiles

### Open, Unencrypted

```
Local $Profile[1]
$Profile[0] = "OPEN"
```

### Open, Unencrypted

```
Local $Profile[11]
$Profile[0] = "OPEN"
$Profile[1] = "Infrastructure"
$Profile[2] = "Automatic"
$Profile[3] = "Open"
$Profile[4] = "Unencrypted"
$Profile[5] = "802.1x Disabled"
$Profile[6] = "No Key Material"
$Profile[7] = "No Key Material"
$Profile[8] = "No Key Index"
$Profile[9] = "No Blob"
$Profile[10] = "No Blob"
```

### Ad Hoc, Open, WEP

```
Local $Profile[9]
$Profile[0] = "ADHOC"
$Profile[1] = "Ad Hoc"
$Profile[4] = "WEP"
$Profile[6] = "Network Key"
$Profile[7] = "Pass!"
$Profile[8] = "2"
```

### Ad Hoc, Shared, WEP

```
Local $Profile[9]
$Profile[0] = "ADHOC"
$Profile[1] = "Ad Hoc"
$Profile[2] = "Automatic"
$Profile[3] = "Shared Key"
$Profile[4] = "WEP"
$Profile[5] = "802.1x Disabled"
$Profile[6] = "Network Key"
$Profile[7] = "Password13chr"
$Profile[8] = "I"
```

### Ad Hoc, Shared, Unencrypted

```
Local $Profile[9]
$Profile[0] = "ADHOC"
$Profile[1] = "Ad Hoc"
$Profile[3] = "Shared Key"
$Profile[6] = "Network Key"
$Profile[7] = "Pass!"
$Profile[8] = I
```

### Shared, WEP, Manual

```
Local $Profile[9]
$Profile[0] = "WEP"
$Profile[2] = "Manual"
$Profile[3] = "Shared Key"
$Profile[4] = "WEP"
$Profile[6] = "Network Key"
$Profile[7] = "Pass!"
$Profile[8] = 4
```

### WPA2-PSK, AES

```
Local $Profile[9]
$Profile[0] = "WPA2PSK"
$Profile[1] = "InRastRuctuRe"
$Profile[2] = "aUtomAtic"
$Profile[3] = "wPA2-PsK"
$Profile[4] = "AeS"
$Profile[5] = "802.1x DiSabLed"
$Profile[6] = "PaSS PhrAse"
$Profile[7] = "Password"
$Profile[8] = "no KEy InDeX"
```

### WPA2-PSK, AES

```
Local $Profile[8]
$Profile[0] = "WPA2PSK"
$Profile[3] = "WPA2-PSK"
$Profile[4] = "AES"
$Profile[6] = "Pass Phrase"
$Profile[7] = "Password"
```

### WPA-PSK, TKIP

```
Local $Profile[8]
$Profile[0] = "WPAPSK"
$Profile[3] = "WPA-PSK"
$Profile[4] = "TKIP"
$Profile[6] = "Pass Phrase"
$Profile[7] = "Password"
```

### WPA-PSK, TKIP

```
Local $Profile[1][8]
$Profile[0][0] = "WPAPSK"
$Profile[0][3] = "WPA-PSK"
$Profile[0][4] = "TKIP"
$Profile[0][6] = "Pass Phrase"
$Profile[0][7] = "Password"
```

### WPA-PSK, AES, Manual

```
Local $Profile[8]
$Profile[0] = "WPAPSK"
$Profile[2] = "Manual"
$Profile[3] = "WPA-PSK"
$Profile[4] = "AES"
$Profile[6] = "Network Key"
$Profile[7] = "4232726FB4" & _
"F14DEEC4D686372EEB30" & _
"20DDBCD9F7503719EDF" & _
"01B668D4E0D0355"
```

## TLS

```
Local $Profile[3][6]
$Profile[0][0] = "WPA2TLS"
$Profile[0][3] = "WPA2"
$Profile[0][4] = "AES"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "TLS"
$Profile[1][1] = "Certificate"
$Profile[1][2] = "Simple Selection"
$Profile[1][3] = "Same User Name"
$Profile[2][0] = "Validate Certificate"
$Profile[2][1] = "Don't Prompt User"
$Profile[2][2] = "^.*\microsoft\.com$"
$Profile[2][3] = "cb a1 c5 f8 b0 e3 5e b8" & _
" b9 45 12 d3 f9 34 a2 e9 06 10 d3 36"
$Profile[2][4] = "d1 eb 23 a4 6d 17 d6 8f" & _
" d9 25 64 c2 f1 f1 60 17 64 d8 e3 49"
```

## TLS

```
Local $Profile[2][6]
$Profile[0][0] = "WPATLS"
$Profile[0][3] = "WPA"
$Profile[0][4] = "TKIP"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "TLS"
$Profile[1][1] = "Certificate"
$Profile[1][2] = "No Simple Selection"
$Profile[1][3] = "Same User Name"
```

## TLS

```
Local $Profile[11]
$Profile[0] = "WPATLS"
$Profile[3] = "WPA"
$Profile[4] = "TKIP"
$Profile[5] = "802.1x Enabled"
$Profile[9] = "TLS"
$Profile[10] = "020000002A00000007000" & _
"00000000000000000000000000000000" & _
"00000000000000000000000000000000"
```

## TLS

```
Local $Profile[3][6]
$Profile[0][0] = "WPA2TLS"
$Profile[0][3] = "WPA2"
$Profile[0][4] = "AES"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "Validate Certificate"
$Profile[1][3] = "cba1c5f8b0e35eb8" & _
" b94512d3f934a2e90610d336"
$Profile[2][0] = "TLS"
$Profile[2][1] = "Smart Card"
$Profile[2][3] = "Different User Name"
```

## PEAP-MSCHAP

```
Local $Profile[4][6]
$Profile[0][0] = "WPAPEAPMSCHAP"
$Profile[0][3] = "WPA"
$Profile[0][4] = "AES"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "PEAP"
$Profile[1][1] = "Fast Reconnect"
$Profile[1][2] = "Quarantine Checks"
$Profile[1][3] = "Require Cryptobinding"
$Profile[2][0] = "Validate Certificate"
$Profile[2][1] = "Don't Prompt User"
$Profile[2][2] = "^.*\microsoft\.com$"
$Profile[3][0] = "MSCHAP"
$Profile[3][1] = "Use Win Logon"
```

## PEAP-MSCHAP

```
Local $Profile[4][6]
$Profile[0][0] = "WPA2PEAPMSCHAP"
$Profile[0][3] = "WPA2"
$Profile[0][4] = "AES"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "PEAP"
$Profile[1][1] = "No Fast Reconnect"
$Profile[1][2] = "No Quarantine Checks"
$Profile[1][3] = "Don't Require Cryptobinding"
$Profile[2][0] = "Validate Certificate"
$Profile[3][0] = "MSCHAP"
$Profile[3][1] = "Don't Use Win Logon"
```

## PEAP-MSCHAP

```
Local $Profile[1]
$Profile[0] = "WPA2PEAPMSCHAP"
$Profile[3] = "WPA2"
$Profile[4] = "AES"
$Profile[5] = "802.1x Enabled"
$Profile[9] = "PEAP"
$Profile[10] = "010000003E00000010000" & _
"000000000010000001500000015000000" & _
"00000000000001000000170000001A0000" & _
"0001000000000000000000000000000000000000"
```

## PEAP-TLS

```
Local $Profile[5][6]
$Profile[0][0] = "WPA2PEAPTLS"
$Profile[0][3] = "WPA2"
$Profile[0][4] = "AES"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "PEAP"
$Profile[1][1] = "Fast Reconnect"
$Profile[1][2] = "Quarantine Checks"
$Profile[1][3] = "Don't Require Cryptobinding"
$Profile[2][0] = "Validate Certificate"
$Profile[2][1] = "Prompt User"
$Profile[2][4] = "d1 eb 23 a4 6d 17 d6 8f" & _
"d9 25 64 c2 f1 f1 60 17 64 d8 e3 49"
$Profile[3][0] = "TLS"
$Profile[3][1] = "Certificate"
$Profile[3][2] = "Simple Selection"
$Profile[3][3] = "Same User Name"
$Profile[4][0] = "Validate Certificate"
$Profile[4][2] = "^.*\.microsoft\.com$"
$Profile[4][3] = "cb a1 c5 f8 b0 e3 5e b8" & _
"b9 45 12 d3 f9 34 a2 e9 06 10 d3 36"
```

## PEAP-TLS

```
Local $Profile[3][6]
$Profile[0][0] = "WPAPEAPTLS"
$Profile[0][3] = "WPA"
$Profile[0][4] = "AES"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "PEAP"
$Profile[1][1] = "Fast Reconnect"
$Profile[1][2] = "No Quarantine Checks"
$Profile[1][3] = "Require Cryptobinding"
$Profile[2][0] = "TLS"
$Profile[2][1] = "Certificate"
$Profile[2][2] = "Simple Selection"
$Profile[2][3] = "Same User Name"
```

## PEAP-TLS

```
Local $Profile[4][6]
$Profile[0][0] = "WPAPEAPTLS"
$Profile[0][3] = "WPA"
$Profile[0][4] = "AES"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "PEAP"
$Profile[1][1] = "Fast Reconnect"
$Profile[1][2] = "No Quarantine Checks"
$Profile[1][3] = "Don't Require Cryptobinding"
$Profile[2][0] = "Validate Certificate"
$Profile[2][1] = "Prompt User"
$Profile[2][2] = "^.*\.microsoft\.com$"
$Profile[2][4] = "d1 eb 23 a4 6d 17 d6 8f" & _
"d9 25 64 c2 f1 f1 60 17 64 d8 e3 49"
$Profile[3][0] = "TLS"
$Profile[3][1] = "Certificate"
$Profile[3][2] = "Simple Selection"
$Profile[3][3] = "Same User Name"
```

## PEAP-TLS

```
Local $Profile[5][6]
$Profile[0][0] = "WPA2PEAPTLS"
$Profile[0][3] = "WPA2"
$Profile[0][4] = "AES"
$Profile[0][5] = "802.1x Enabled"
$Profile[1][0] = "PEAP"
$Profile[1][1] = "Fast Reconnect"
$Profile[1][2] = "Quarantine Checks"
$Profile[1][3] = "Don't Require Cryptobinding"
$Profile[2][0] = "TLS"
$Profile[2][1] = "Certificate"
$Profile[2][2] = "Simple Selection"
$Profile[2][3] = "Same User Name"
$Profile[3][0] = "Don't Validate Certificate"
$Profile[4][0] = "Validate Certificate"
$Profile[4][1] = "Prompt User"
$Profile[4][2] = "^.*\.microsoft\.com$"
```

## Functions

### **\_Wlan\_StartSession()**

|                   |                                                                                                                         |
|-------------------|-------------------------------------------------------------------------------------------------------------------------|
| Description:      | Calls <code>_Wlan_OpenHandle()</code> , <code>_Wlan_EnumInterfaces()</code> and <code>_Wlan_SetGlobalConstants()</code> |
| Version a Syntax: | <code>_Wlan_StartSession()</code>                                                                                       |
| Version b Syntax: | <code>_Wlan_StartSession()</code>                                                                                       |

#### *Return Value:*

`$InterfaceArray[$n][0]` – Pointer to the interface's GUID (`$pGUID`)  
`$InterfaceArray[$n][1]` – The name of the interface  
`$InterfaceArray[$n][2]` – Connection status

`@extended` is set to the newly obtained client handle (`$hClientHandle`)

#### *Possible Values:*

Connection status (`InterfaceArray[$n][2]`)  
"Connected"  
"Disconnected"  
"Authenticating"

#### *Remarks:*

The default `$pGUID` value is set to `$InterfaceArray[0][0]`  
The default `$ hClientHandle` value is set to `@extended`

---

### **\_Wlan\_EndSession()**

|                   |                                                                            |
|-------------------|----------------------------------------------------------------------------|
| Description:      | Calls <code>_Wlan_CloseHandle()</code> and closes <code>WlanAPI.dll</code> |
| Version a Syntax: | <code>_Wlan_EndSession(\$hClientHandle)</code>                             |
| Version b Syntax: | <code>_Wlan_EndSession(\$hClientHandle)</code>                             |

#### *Remarks:*

This function should be placed in the AutoIt exit function (default `OnAutoItExit()`).

---

### **\_Wlan\_OpenHandle()**

|                   |                                                   |
|-------------------|---------------------------------------------------|
| Description:      | Opens a new client handle for following functions |
| Version a Syntax: | <code>_Wlan_OpenHandle()</code>                   |
| Version b Syntax: | <code>_Wlan_OpenHandle()</code>                   |

#### *Return Value:*

`$hClientHandle` – Client handle for subsequent functions

#### *Remarks:*

This function is superseded by `_Wlan_StartSession()`.



### **\_Wlan\_CloseHandle()**

|                   |                                                   |
|-------------------|---------------------------------------------------|
| Description:      | Closes a client handle                            |
| Version a Syntax: | <code>_Wlan_CloseHandle([\$hClientHandle])</code> |
| Version b Syntax: | <code>_Wlan_CloseHandle(\$hClientHandle)</code>   |

#### *Remarks:*

This function is superseded by `_Wlan_EndSession()`.

---

### **\_Wlan\_EnumInterfaces()**

|                   |                                                              |
|-------------------|--------------------------------------------------------------|
| Description:      | Enumerates and gathers interface information (incl. \$pGUID) |
| Version a Syntax: | <code>_Wlan_EnumInterfaces([\$hClientHandle])</code>         |
| Version b Syntax: | <code>_Wlan_EnumInterfaces(\$hClientHandle)</code>           |

#### *Return Value:*

\$InterfaceArray[\$n][0] – Pointer to the interface's GUID (\$pGUID)

\$InterfaceArray[\$n][1] – The name of the interface

\$InterfaceArray[\$n][2] – Connection status

#### *Possible Values:*

Connection status (InterfaceArray[\$n][2])

"Connected"

"Disconnected"

"Authenticating"

#### *Remarks:*

This function is superseded by `_Wlan_StartSession()` and `_Wlan_QueryInterface()`.

---

### **\_Wlan\_SetGlobalConstants()**

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| Description:      | Sets default \$pGUID and \$hClientHandle values                   |
| Version a Syntax: | <code>_Wlan_SetGlobalConstants(\$pGUID[, \$hClientHandle])</code> |
| Version b Syntax: | <code>_Wlan_SetGlobalConstants(\$hClientHandle, \$pGUID)</code>   |

#### *Remarks:*

A blank string ("" ) can be used for either parameter to preserve the existing default value.

---

### **\_Wlan\_Scan()**

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| Description:      | Initiates a scan for wireless networks                |
| Version a Syntax: | <code>_Wlan_Scan([\$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_Scan(\$hClientHandle, \$pGUID)</code>     |

#### *Remarks:*

Use `_Wlan_GetAvailableNetworkList()` to gather information on available networks.

## \_Wlan\_GetAvailableNetworkList()

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| Description:      | Retrieves the list of available networks on a given interface          |
| Version a Syntax: | _Wlan_GetAvailableNetworkList([\$dwFlag[, \$pGUID[, \$hClientHandle]]) |
| Version b Syntax: | _Wlan_GetAvailableNetworkList(\$hClientHandle, \$pGUID, \$dwFlag)      |

### *Parameters:*

\$dwFlag

- 0 – infrastructure only
- 1 – include ad hoc networks
- 2 – include hidden networks

### *Return Value:*

\$AvailableNetworkArray[\$n][0] - SSID  
\$AvailableNetworkArray[\$n][1] - Network type  
\$AvailableNetworkArray[\$n][2] - Connectability  
\$AvailableNetworkArray[\$n][3] - Signal strength  
\$AvailableNetworkArray[\$n][4] - Authentication method  
\$AvailableNetworkArray[\$n][5] - Encryption method  
\$AvailableNetworkArray[\$n][6] - Profile status  
\$AvailableNetworkArray[\$n][7] - Not connectable reason

### *Possible Value:*

Network type (InterfaceArray[\$n][1])  
"Infrastructure"  
"Ad Hoc"

Connectability (\$AvailableNetworkArray[\$n][2])  
"Connectable"  
"Not Connectable"

Signal strength (\$AvailableNetworkArray[\$n][3])  
0 to 100 (scale is linear)  
0 = -100dbm or less  
100 = -50dbm or more

Authentication method (\$AvailableNetworkArray[\$n][4])  
"Open"  
"Shared Key"  
"WPA"  
"WPA-PSK"  
"WPA2"  
"WPA2-PSK"

Encryption method (\$AvailableNetworkArray[\$n][5])  
"Unencrypted"  
"WEP"  
"TKIP"  
"AES"

Profile status (\$AvailableNetworkArray[\$n][6])  
"No Profile"  
"Profile"  
"Connected"

*Remarks:*

If the connectability value for a network is "Connectable", the Not connectable reason value will be 0.

The default for \$dwflag in version a is 0

---

### **\_Wlan\_Connect()**

|                   |                                                         |
|-------------------|---------------------------------------------------------|
| Description:      | Connects to a network if a corresponding profile exists |
| Version a Syntax: | _Wlan_Connect(\$SSID[, \$pGUID[, \$hClientHandle]])     |
| Version b Syntax: | _Wlan_Connect(\$hClientHandle, \$pGUID, \$SSID)         |

*Remarks:*

If a higher priority network is in range (and the corresponding profile is set to automatic), WZC will eventually try to connect to it in favour of the current connection.

If a connection fails, WZC will work down the list of automatic profiles.

---

### **\_Wlan\_Disconnect()**

|                   |                                                |
|-------------------|------------------------------------------------|
| Description:      | Disconnects an interface from a network        |
| Version a Syntax: | _Wlan_Disconnect([\$pGUID[, \$hClientHandle]]) |
| Version b Syntax: | _Wlan_Disconnect(\$hClientHandle, \$pGUID)     |

---

### **\_Wlan\_ConnectWait()**

|                   |                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------|
| Description:      | Connects to a network if a corresponding profile exists and waits for the connection to complete |
| Version a Syntax: | _Wlan_ConnectWait(\$SSID[, \$Timeout[, \$pGUID[, \$hClientHandle]])                              |
| Version b Syntax: | _Wlan_ConnectWait(\$hClientHandle, \$pGUID, \$SSID, \$Timeout)                                   |

*Return Value:*

The SSID of the connected network

*Remarks:*

The default \$Timeout value in version a is 15

This function may be unreliable with EAP networks. An alternative is to use *Ping()* in a While loop.

---

### **\_Wlan\_DisconnectWait()**

|                   |                                                                  |
|-------------------|------------------------------------------------------------------|
| Description:      | Disconnects from a network and waits for the connection to cease |
| Version a Syntax: | _Wlan_DisconnectWait([\$Timeout[, \$pGUID[, \$hClientHandle]])   |
| Version b Syntax: | _Wlan_DisconnectWait(\$hClientHandle, \$pGUID, \$Timeout)        |

*Return Value:*

The SSID of the connected network

*Remarks:*

The default \$Timeout value in version a is 5

### **\_Wlan\_WaitForDisconnect()**

|                   |                                                                                |
|-------------------|--------------------------------------------------------------------------------|
| Description:      | Idles until disconnected                                                       |
| Version a Syntax: | <code>_Wlan_WaitForDisconnect([\$Timeout[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_WaitForDisconnect(\$hClientHandle, \$pGUID, \$Timeout)</code>      |

#### *Remarks:*

If the \$Timeout value is set to -1 then the function will not time out

The default \$Timeout value in version a is -1

---

### **\_Wlan\_GetProfile()**

|                   |                                                                     |
|-------------------|---------------------------------------------------------------------|
| Description:      | Returns information about an existing profile                       |
| Version a Syntax: | <code>_Wlan_GetProfile(\$SSID[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_GetProfile(\$hClientHandle, \$pGUID, \$SSID)</code>     |

#### *Return Value:*

A 1 dimension, 11 element profile array

#### *Remarks:*

See from Profile Array Format (page 7)

---

### **\_Wlan\_SetProfile()**

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| Description:      | Creates a new profile                                                  |
| Version a Syntax: | <code>_Wlan_SetProfile(\$Profile[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_SetProfile(\$hClientHandle, \$pGUID, \$Profile)</code>     |

#### *Remarks:*

See from Profile Array Format (page 7)

---

### **\_Wlan\_GetProfileXML()**

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| Description:      | Returns information about an existing profile in XML format            |
| Version a Syntax: | <code>_Wlan_GetProfileXML(\$SSID[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_GetProfileXML(\$hClientHandle, \$pGUID, \$SSID)</code>     |

#### *Return Values*

\$Profile – A profile in XML format

---

### **\_Wlan\_SetProfileXML()**

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| Description:      | Creates a new profile from XML format                                     |
| Version a Syntax: | <code>_Wlan_SetProfileXML(\$Profile[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_SetProfileXML(\$hClientHandle, \$pGUID, \$Profile)</code>     |

#### *Remarks:*

See [http://msdn.microsoft.com/en-us/library/aa369853\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa369853(VS.85).aspx) for samples (not all are compatible with XP)

### **\_Wlan\_DeleteProfile()**

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| Description:      | Purges an existing profile                                             |
| Version a Syntax: | <code>_Wlan_DeleteProfile(\$SSID[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_DeleteProfile(\$hClientHandle, \$pGUID, \$SSID)</code>     |

### **\_Wlan\_GetProfileList()**

|                   |                                                                 |
|-------------------|-----------------------------------------------------------------|
| Description:      | Returns an array of profiles in order of priority               |
| Version a Syntax: | <code>_Wlan_GetProfileList([\$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_GetProfileList(\$hClientHandle, \$pGUID)</code>     |

#### *Return Values*

\$aProfileNames – A 1 dimensional array of profile names in order of priority

### **\_Wlan\_SetProfileList()**

|                   |                                                                                  |
|-------------------|----------------------------------------------------------------------------------|
| Description:      | Sets existing profiles in order of priority                                      |
| Version a Syntax: | <code>_Wlan_SetProfileList(\$aProfileNames[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_SetProfileList(\$hClientHandle, \$pGUID, \$aProfileNames)</code>     |

#### *Parameters*

\$aProfileNames – A 1 dimensional array of profile names in order of priority

### **\_Wlan\_SetProfilePosition()**

|                   |                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------|
| Description:      | Changes the priority of a nominated profile                                               |
| Version a Syntax: | <code>_Wlan_SetProfilePosition(\$SSID, \$dwPosition[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_SetProfilePosition(\$hClientHandle, \$pGUID, \$SSID, \$dwPosition)</code>     |

#### *Parameters*

\$dwPosition – Position in the list (0 based)

### **\_Wlan\_QueryInterface()**

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| Description:      | Returns various interface settings or connection information              |
| Version a Syntax: | <code>_Wlan_QueryInterface(\$dwFlag[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_QueryInterface(\$hClientHandle, \$pGUID, \$dwFlag)</code>     |

#### *Parameters*

\$dwFlag

- 0 – Auto config state
- 1 – Accessible network types
- 2 – Connection status
- 3 – Connection information

### *Possible Return Value*

If \$dwFlag = 0  
    "Auto Config Enabled"  
    "Auto Config Disabled"

If \$dwFlag = 1  
    "Infrastructure"  
    "Ad Hoc Only"  
    "Any Available Network"

If \$dwFlag = 2  
    "Connected"  
    "Disconnected"  
    "Authenticating"

If \$dwFlag = 3  
    \$ConnectionAttributes[0] – Connection status  
    \$ConnectionAttributes[1] – SSID  
    \$ConnectionAttributes[2] – Host MAC address  
    \$ConnectionAttributes[3] – Signal strength  
    \$ConnectionAttributes[4] – Security status  
    \$ConnectionAttributes[5] – 802.1x status  
    \$ConnectionAttributes[6] – Authentication  
    \$ConnectionAttributes[7] – Encryption

### *Possible Values*

Connection status (\$ConnectionAttributes[0])  
    "Connected"  
    "Disconnected"  
    "Authenticating"

Signal strength (\$ConnectionAttributes[3])  
    0 to 100 (scale is linear)  
    0 = -100dbm or less  
    100 = -50dbm or more

802.1x status (\$ConnectionAttributes[5])  
    "802.1x Enabled"  
    "802.1x Disabled"

Authentication (\$ConnectionAttributes[6])  
    "Open"  
    "Shared Key"  
    "WPA"  
    "WPA-PSK"  
    "WPA2"  
    "WPA2-PSK"

Encryption (\$ConnectionAttributes[7])  
    "Unencrypted"  
    "WEP"  
    "TKIP"  
    "AES"

### **\_Wlan\_SetInterface()**

|                   |                                                                                    |
|-------------------|------------------------------------------------------------------------------------|
| Description:      | Sets various interface parameters                                                  |
| Version a Syntax: | <code>_Wlan_SetInterface(\$dwFlag, \$strData[, \$pGUID[, \$hClientHandle]])</code> |
| Version b Syntax: | <code>_Wlan_SetInterface(\$hClientHandle, \$pGUID, \$dwFlag, \$strData)</code>     |

#### *Parameters*

**\$dwFlag**  
0 – Auto config state  
    **\$StrData**  
        "Auto Config Enabled"  
        "Auto Config Disabled"  
1 – Accessible network types  
    **\$StrData**  
        "Infrastructure"  
        "Ad Hoc Only"  
        "Any Available Network"

---

### **\_Wlan\_GenerateXMLProfile()**

|                   |                                                  |
|-------------------|--------------------------------------------------|
| Description:      | Generates an XML profile from an array format    |
| Version a Syntax: | <code>_Wlan_GenerateXMLProfile(\$Profile)</code> |
| Version b Syntax: | <code>_Wlan_GenerateXMLProfile(\$Profile)</code> |

#### *Return Values*

**\$Profile** – A profile in XML format

---

### **\_Wlan\_StringTopGUID()**

|                   |                                             |
|-------------------|---------------------------------------------|
| Description:      | Returns a \$pGUID value from a GUID string  |
| Version a Syntax: | <code>_Wlan_StringTopGuid(\$strGUID)</code> |
| Version b Syntax: | <code>_Wlan_StringTopGuid(\$strGUID)</code> |

#### *Remarks*

It is now possible to create multiple \$pGUID values from this function without invalidating previous values.

---

### **\_Wlan\_pGUIDToString()**

|                   |                                             |
|-------------------|---------------------------------------------|
| Description:      | Returns a GUID string from a \$pGUID value  |
| Version a Syntax: | <code>_Wlan_pGuidToString([\$pGUID])</code> |
| Version b Syntax: | <code>_Wlan_pGuidToString(\$pGUID)</code>   |