WinPcap Autoit3 UDF

v1.2c (updated: May 23rd 2011)

http://opensource.grisambre.net/pcapau3

[Examples] [Function Reference] [Download] [History] [Links]

The below script (UDF) allows very simply from an Autoit script to access the main functionalities offered by the WinPcap driver: capture, filter, save/read and send data packets on a network interface. This was developped with Autoit3 v3.3.0.0 and is free and "open source", and licensed under the GNU GPL 3 - copyleft Nicolas Ricquemaque 2008 [contact: opensource (arobase) grisambre dot net].

Quick examples

A few quick examples is the best way to show how it works ! However, only minimal error detection is made here. For a more comprehensive example, just have a look into the **winpcap_demo.au3** included with the library archive.

Example(1): Displaying your device list with full information:

Code:

```
#include <Array.au3>
```

```
#include <Winpcap.au3>

$winpcap=_PcapSetup()     ;initialize winpcap
$pcap_devices=_PcapGetDeviceList()   ;get devices list
_ArrayDisplay($pcap_devices,"Devices list",-1,1) ;display it
_PcapFree() ;close winpcap
```

Example(2): Capturing ICMP packets for 10 seconds

Code:

```
;initialise the Library
$winpcap=_PcapSetup()
If ($winpcap=-1) Then
    MsgBox(16,"Pcap error !","WinPcap not found !")
    exit
EndIf

;Get the interfaces list for which a capture is possible
$pcap_devices=_PcapGetDeviceList()
If ($pcap_devices=-1) Then
    MsgBox(16,"Pcap error !",_PcapGetLastError())
    exit
EndIf

;Start a capture on interface #0, for ICMP packets only
$pcap=_PcapStartCapture($pcap_devices[0][0],"icmp")
If ($pcap=-1) Then
    MsgBox(16,"Pcap error !",_PcapGetLastError())
EndIf

;Detect of what type is the opened interface (ethernet, ATM, X25...)
```

```
$linktype=_PcapGetLinkType($pcap)
If ($linktype[1]<>"EN10MB") Then
    MsgBox(16,"Pcap error !","This example only accepts Ethernet devices...")
Endif

; Capture anything that matches our filter "ICMP" for 10 seconds...
$time0=TimerInit()
While (TimerDiff($time0)<10000) ; capture the packets for 10 seconds...
    $packet=_PcapGetPacket($pcap)
    If IsArray($packet) Then
        ; here do something with your data
    EndIf
Wend

; Stop capture
_PcapStopCapture($pcap)

; release ressources
_PcapFree()
```

Example(3): Saving http traffic to a pcap file for 10s...

Code:

```
$winpcap=_PcapSetup() ; initialise the Library
$pcap_devices=_PcapGetDeviceList() ; Get the interfaces list for which a capture is possible

; Start a capture on interface #0, in promiscuous mode, for http packets only
$pcap=_PcapStartCapture($pcap_devices[0][0],"tcp port 80",1)

; Open pcap file for writting
```

```
$pcapfile=_PcapSaveToFile($pcap,"mycapture.pcap")
If ($pcapfile=0) Then MsgBox(16,"Pcap error !",_PcapGetLastError())

; Write all http traffic to the file for 10s...
$time0=TimerInit()
While (TimerDiff($time0)<10000)
    $packet=_PcapGetPacket($pcap)
    If IsArray($packet) Then _PcapWriteLastPacket($pcapfile)
Wend

_PcapStopCaptureFile($pcapfile) ; Close pcap file
_PcapStopCapture($pcap) ; Stop capture
_PcapFree() ; release ressources
```

Example(4): Reading a whole existing pcap file...

Code:

```
$winpcap=_PcapSetup() ; initialise the Library

; Open pcap file for reading
$pcap=_PcapStartCapture("file://mycapture.pcap")

; Read whatever is in the file until its end.
Do
    $packet=_PcapGetPacket($pcap)
    If IsArray($packet) Then
        ; Do something with your data here...
    EndIf
Until $packet=-2  ; EOF

_PcapStopCapture($pcap) ; Stop capture
```

```
_PcapFree() ; release ressources
```

Example(5): Sending a valid ethernet broadcast on your lan...

Code:

```
#include <Winpcap.au3>

$winpcap=_PcapSetup()    ; initialize winpcap
$pcap_devices=_PcapGetDeviceList()   ; get devices list
$pcap=_PcapStartCapture($pcap_devices[1][0]) ; my interface

$broadcastmac="FFFFFFFFFFFF" ; broacast
$mymac=StringReplace($pcap_devices[1][6],":","") ; my mac address in hex
$ethertype="3366"    ; fake ethertype, means nothing, just for example...
$mydata="0123456789"    ; dumb padding...

$mypacket="0x"&$broadcastmac&$mymac&$ethertype&$mydata ; stick together to a binary string !
_PcapSendPacket($pcap,$mypacket) ; sends a valid ethernet broadcast !

_PcapFree() ; close winpcap
```

UDF Functions reference

_PcapSetup()

Initialise the Winpcap DLL and setup some Global variables.

Parameters: *None*

Return Value:

- **On success:** a string containing the complete winpcap version information
- **On failure:** -1 (Winpcap is probably not installed)

_PcapFree()

Free resources opened by a previous call to *_PcapSetup()*.

Parameters: *None*

Return Value: *None*

_PcapGetLastError([$pcap=0])

Function to be called to get clues why an error was returned by any other function in this library.

Parameters:

- **$pcap** (optional) is a capture handler (as returned by a call to _PcapStartCapture()).

Return Value:

- A string containing (or not) the description for the last error.

_PcapGetDeviceList()

Returns a list of interface/devices which can be opened for capture.

Parameters: *None*

Return Value:

- **On success:** a 2D array containing the device list information. For each device:
  - [x][0]=(string) Device Name for device x (which will be given in call to _PcapStartCapture())
  - [x][1]=(string) Description for device x
  - [x][2]=(int) Linktype (known as DLT, see winpcap documentation or pcap-bpf.h for details)
  - [x][3]=(string) Linktype as text (see winpcap documentation or pcap-bpf.h for details)
  - [x][4]=(string) Linktype description
  - [x][5]=(int) Link Speed in bits per second
  - [x][6]=(string) MAC address, if available
  - [x][7]=(string) IPv4 address, if available
  - [x][8]=(string) IPv4 netmask, if available
  - [x][9]=(string) IPv4 broadcast, if available
  - [x][10]=(string) IPv6 address, if available
  - [x][11]=(string) IPv6 netmask, if available
  - [x][12]=(string) IPv6 broadcast, if available
  - [x][13]=Flags for device (currently, the only possible flag is PCAP_IF_LOOPBACK which has the value 1, meaning the device is a loopback)
- **On failure:** -1 (No capture device found ?)

_PcapGetLinkType($pcap)

Provides LinkType for opened capture $pcap.

Parameters:

- **$pcap** is a capture handler (as returned by a call to _PcapStartCapture()).

Return Value:

- **On success:** an array with some linktype information:
  - [0]: (int) value of link type
  - [1] (string) name of linktype
  - [2] (string) description of linktype
- **On failure:** -1

_PcapGetStats($pcap)

Provide some statistics about the current capture.

Parameters:

- **$pcap** is a capture handler (as returned by a call to _PcapStartCapture()).

Return Value:

- **On success:** a 2D array with some capture statistics:
  - [0][0]: (int) number of Packets received by Interface

    - [0][1]: (string) "Packets received by Interface"
    - [1][0] (int) number of Packets dropped by WinPcap
    - [1][1] (string) "Packets dropped by WinPcap"
    - [2][0] (int) number of Packets dropped by Interface
    - [2][1] (string) "Packets dropped by Interface"
    - [3][0] (int) number of Packets captured
    - [3][1] (string) "Packets captured"
    - [4][0] (int) total number Bytes in packets captured
    - [4][1] (string) "Bytes in packets captured"
    - [5][0] (int) number mS since capture start
    - [5][1] (string) "mS since capture start"
- **On failure:** -1

_PcapStartCapture($DeviceName[,$filter=""[,$promiscuous=0[,$PacketLen=65536[,$buffersize=0[,$realtime=1]]]]])

Starts a non-blocking capture on interface $DeviceName.

Parameters:

- **$DeviceName** A string giving the devide to open (as returned by a call to _PcapGetDeviceList()). If given as "file://pathtofile.pcap" will also open a pcap capture file.
- **$filter** (optional) string of a pcap filter expression (see http://www.winpcap.org/docs/docs_40_2/html/group__language.html). By default, no filter is applied.
- **$promiscuous** (optional) put 1 to make the capture "promiscuous" (interface will record packets that is not directed directly at it). By default is 0 (no).
- **$PacketLen** (optional) An int giving the maximal part of each packet that will be captured. By default, the value is 65536.
- **$buffersize** (optional) int giving the size of the buffer Winpcap should allow to store the traffic. If 0 it uses default winpcap buffer size, 1MB.

- **$realtime** (optional) Reads driver data in realtime (as soon a a packet is sent/received, it becomes available for reading. It gives RealTime information, but can affect performance badly). By default, true. If false, the driver transfers his data for a minimum amount of 16kB or every second.

Return Value:

- **On success:** a Ptr to a pcap handler.
- **On failure:** -1

_PcapStopCapture($pcap)

Stops an previously opened capture.

Parameters:

- **$pcap** is a capture handler (as returned by a call to _PcapStartCapture()).

Return Value: *None*

_PcapIsPacketReady($pcap)

Returns true if some packets has been received and is ready for reading.

Parameters:

- **$pcap** is a capture handler (as returned by a call to _PcapStartCapture()).

Return Value:

- **On success:** *true* (at least one packet is in buffer)
- **On failure:** *false* (nothing in buffer)

_PcapGetPacket($pcap)

Get last packet captured from Winpcap buffer.

Parameters:

- **$pcap** is a capture handler (as returned by a call to _PcapStartCapture()).

Return Value:

- **On success:** an array with some packet information and Data:
  - [0]: (string) Time the packet was received (format hh:mm:ss.usus us)
  - [1]: (int) Captured length
  - [2]: (int) Packet length
  - [3]: (binary) Packet Data
- **On failure:** an int giving the reason why no packet was received:
  - 0 : nothing received
  - -1 : error reading
  - -2 : EOF (in case the capture device is a pcap file)

_PcapSendPacket($pcap,$data)

Sends a raw packet to the interface.

Parameters:

- **$pcap** is a capture handler (as returned by a call to _PcapStartCapture()).
- **$data** is a binary string containing the packet to send.

Return Value:

- **On success:** 0
- **On failure:** -1

_PcapSaveToFile($pcap,$filename)

Opens a pcap file so save packets.

Parameters:

- **$pcap** is a capture handler (as returned by a call to _PcapStartCapture()).
- **$filename** string containing the path to the file to save to.

Return Value:

- **On success:** A Ptr to the pcapfile handler.
- **On failure:** -1

_PcapWriteLastPacket($handle)

Writes the last received packet to the pcap file previously opened by a call to *_PcapSaveToFile()*.

Parameters:

- **$handle** is pcapfile handler (as returned by a call to *_PcapSaveToFile()*).

Return Value:

- **On success:** Nothing
- **On failure:** -1

_PcapStopCaptureFile($handle)

Closes the pcap file previously opened by a call to *_PcapSaveToFile()*.

Parameters:

- **$handle** is pcapfile handler (as returned by a call to *_PcapSaveToFile()*).

Return Value:

- **On success:** Nothing
- **On failure:** -1

_PcapListLinkTypes($pcap)

Get a list of available LinkTypes for opened capture $pcap.

Parameters:

- **$pcap** is a capture handler (as returned by a call to _PcapStartCapture()).

Return Value:

- **On success:** a 2D array with some linktype information, for each possible linktype:
    - [n][0]: (int) value of link type
    - [n][1] (string) name of linktype
    - [n][2] (string) description of linktype
- **On failure:** -1

_PcapSetLinkType($pcap,$dlt)

Set one of the available linktype given by a call to *_PcapListLinkTypes()* as the active linktype for opened capture $pcap.

Parameters:

- **$pcap** is a capture handler (as returned by a call to *_PcapStartCapture()*).
- **$dlt** is an int giving the linktype to select, as return in field [0] of a call to *_PcapListLinkTypes()*

Return Value:

- **On success:** 0
- **On failure:** -1

_PcapBinaryGetVal($data,$offset,$bytes)

Extract a value from a binary string (from 1 to 4 bytes, so 8 to 32 bits unsigned).

Parameters:

- **$data** is a the binary string to extract data from (for an example: the packet data provided by *_PcapGetPacket()*)
- **$offset** is an int giving the offset from the beginning of the binary string. 1 for first byte.
- **$bytes** is an int between 1 and 4 giving the size in bytes of the value to extract

Return Value:

- An unsigned integer.

_PcapBinarySetVal(Byref $data,$offset,$value,$bytes)

Sets a value inside a binary string (from 1 to 4 bytes, so 8 to 32 bits unsigned). Before calling this function, one should make sure that $data contains at least $offset+$bytes binary bytes !

Parameters:

- **$data** is a the binary string to set the value into (for an example: a new packet beeing forged)
- **$offset** is an int giving the offset from the beginning of the binary string. 1 for first byte.
- **$value** the int value to insert (between 0 and 2^32-1).
- **$bytes** is an int between 1 and 4 giving the size in bytes of the value to insert (so from 8 to 32 bits)

Return Value:

- Nothing.

_PcapIpCheckSum($data,$ipoffset=14)

Computes the IP checksum of the packet; useful for forging a new packet. Before calling this function, one should make sure that $data contains an IP packet !

Parameters:

- **$data** is a the binary string of the IP packet.
- **$ipoffset** is an optional int giving the offset of the IP packet from the beginning of the frame; by default, 14 is assumed (ethernet). 1 for first byte.

Return Value:

- A 32bits unsigned integer giving the IP header checksum value.

_PcapIcmpCheckSum($data,$ipoffset=14)

Computes the ICMP checksum of the packet; useful for forging a new packet. Before calling this function, one should make sure that $data contains an ICMP packet !

Parameters:

- **$data** is a the binary string of the ICMP packet.
- **$ipoffset** is an optional int giving the offset of the IP packet from the beginning of the frame; by default, 14 is assumed (ethernet). 1 for first byte.

Return Value:

- A 32bits unsigned integer giving the ICMP header checksum value.

_PcapTcpCheckSum($data,$ipoffset=14)

Computes the TCP checksum of the packet; useful for forging a new packet. Before calling this function, one should make sure that $data contains a TCP packet !

Parameters:

- **$data** is a the binary string of the TCP packet.
- **$ipoffset** is an optional int giving the offset of the IP packet from the beginning of the frame; by default, 14 is assumed (ethernet). 1 for first byte.

Return Value:

- A 32bits unsigned integer giving the TCP header checksum value.

_PcapUdpCheckSum($data,$ipoffset=14)

Computes the UDP checksum of the packet; useful for forging a new packet. Before calling this function, one should make sure that $data contains an UDP packet !

Parameters:

- **$data** is a the binary string of the UDP packet.
- **$ipoffset** is an optional int giving the offset of the IP packet from the beginning of the frame; by default, 14 is assumed (ethernet). 1 for first byte.

Return Value:

- A 32bits unsigned integer giving the UDP header checksum value.

_PcapCleanDeviceName($fullname)

Remove boring text from the WinPcap device name (example: returns "VIA Rhine II Fast Ethernet Adapter (Microsoft's Packet Scheduler)" instead of "Network adapter 'VIA Rhine II Fast Ethernet Adapter (Microsoft's Packet Scheduler) ' on local host").

Parameters:

- **$fullname** is the string of device name as returned by _PcapGetDeviceList()[0].

Return Value:

- The cleaned string.

Download

Contents :

- *winpcapau3.html* : This quick documentation
- *winpcap.au3* : The UDF itself !
- *winpcap_demo.au3* : A demonstration script for the UDF.
- *licence* : The GNU GPL 3 licence text

Actual version (1.2b): winpcapau3.zip

History

v1.0a (April 2009)
First public release.

v1.0b (April 2009)

- Corrected a memory allocation bug in _PcapGetDeviceList()
- Added functions **_PcapListLinkTypes()** and **_PcapSetLinkType()**

v1.1a (April 11th 2009)

- **_PcapGetDeviceList()** is now providing many more informations (linktype, ipv4 and ipv6 addresses, mac address, linkspeed...)
- Added function **_PcapIsPacketReady()**
- Added option *$realtime* in **_PcapStartCapture()**

v1.2a (April 22th 2009) : A few IP utility functions...

- Added BinaryString manipulation functions **_PcapBinaryGetVal()** and **_PcapBinarySetVal()**.
- Added checksum computation functions; IP: **_PcapIpCheckSum()**, ICMP: **_PcapIcmpCheckSum()**, TCP: **_PcapTcpCheckSum()**, UDP: **_PcapUdpCheckSum()**

v1.2b (April 24th 2009)

- Added function _**PcapCleanDeviceName()**.

---

v1.2c (April 23rd 2011)

- corrected bug in _**PcapStartCapture()** thanks to Wei.

Links

- Winpcap driver
- Autoit3 scripting language.