1 Building display filter expressions

Wireshark provides a simple but powerful display filter language that allows you to build quite complex filter expressions. You can compare values in packets as well as combine expressions into more specific expressions. The following sections provide more information on doing this.

> **Tip!**
>
> You will find a lot of Display Filter examples at the **Wireshark Wiki Display Filter page** at http://wiki.wireshark.org/DisplayFilters.

1.1 Display filter fields

Every field in the packet details pane can be used as a filter string, this will result in showing only the packets where this field exists. For example: the filter string: **tcp** will show all packets containing the tcp protocol.

There is a complete list of all filter fields available through the menu item "Help/Supported Protocols" in the page "Display Filter Fields" of the Supported Protocols dialog.

XXX - add some more info here and a link to the statusbar info.

1.2 Comparing values

You can build display filters that compare values using a number of different comparison operators. They are shown in "Display Filter comparison operators".

> **Tip!**
>
> You can use English and C-like terms in the same way, they can even be mixed in a filter string!

1.2.1 Display Filter comparison operators

English	C-like	Description and example
eq	==	**Equal** ip.src==10.0.0.5
ne	!=	**Not equal** ip.src!=10.0.0.5
gt	>	**Greater than** frame.len > 10
lt	<	**Less than** frame.len < 128
ge	>=	**Greater than or equal to** frame.len ge 0x100
le	<=	**Less than or equal to** frame.len <= 0x20

In addition, all protocol fields are typed. "Display Filter Field Types" provides a list of the types and example of how to express them.

1.2.2 Display Filter Field Types

Type	Example
Unsigned integer (8-bit, 16-bit, 24-bit, 32-bit)	You can express integers in decimal, octal, or hexadecimal. The following display filters are equivalent: ip.len le 1500 ip.len le 02734 ip.len le 0x436
Signed integer (8-bit, 16-bit, 24-bit, 32-bit)	
Boolean	A boolean field is present in the protocol decode only if its value is true. For example, **tcp.flags.syn** is present, and thus true, only if the SYN flag is present in a TCP segment header. Thus the filter expression **tcp.flags.syn** will select only those packets for which this flag exists, that is, TCP segments where the segment header contains the SYN flag. Similarly, to find source-routed token ring packets, use a filter expression of **tr.sr**.
Ethernet address (6 bytes)	Separators can be a colon (:), dot (.) or dash (-) and can have one or two bytes between separators: eth.dst == ff:ff:ff:ff:ff:ff eth.dst == ff-ff-ff-ff-ff-ff eth.dst == ffff.ffff.ffff
IPv4 address	ip.addr == 192.168.0.1 Classless InterDomain Routing (CIDR) notation can be used to test if an IPv4 address is in a certain subnet. For example, this display filter will find all packets in the 129.111 Class-B network: ip.addr == 129.111.0.0/16
IPv6 address	ipv6.addr == ::1
IPX address	ipx.addr == 00000000.ffffffffffff
String (text)	http.request.uri == "http://www.wireshark.org/"

1.3 Combining expressions

You can combine filter expressions in Wireshark using the logical operators shown in "Display Filter Logical Operations"

1.3.1 Display Filter Logical Operations

English	C-like	Description and example
and	&&	**Logical AND** ip.src==10.0.0.5 and tcp.flags.fin
or	||	**Logical OR** ip.scr==10.0.0.5 or ip.src==192.1.1.1
xor	^^	**Logical XOR** tr.dst[0:3] == 0.6.29 xor tr.src[0:3] == 0.6.29
not	!	**Logical NOT** not llc
[...]		**Substring Operator** Wireshark allows you to select subsequences of a sequence in rather elaborate ways. After a label you can place a pair of brackets [] containing a comma separated list of range specifiers. eth.src[0:3] == 00:00:83 The example above uses the n:m format to specify a single range. In this case n is the beginning offset and m is the length of the range being specified. eth.src[1-2] == 00:83 The example above uses the n-m format to specify a single range. In this case n is the beginning offset and m is the ending offset. eth.src[:4] == 00:00:83:00 The example above uses the :m format, which takes everything from the beginning of a sequence to offset m. It is equivalent to 0:m eth.src[4:] == 20:20 The example above uses the n: format, which takes everything from offset n to the end of the sequence. eth.src[2] == 83 The example above uses the n format to specify a single range. In this case the element in the sequence at offset n is selected. This is equivalent to n:1. eth.src[0:3,1-2,:4,4:,2] == 00:00:83:00:83:00:00:83:00:20:20:83 Wireshark allows you to string together single ranges in a comma separated list to form compound ranges as shown above.

1.4 A common mistake

Warning!

Using the != operator on combined expressions like: eth.addr, ip.addr, tcp.port, udp.port and alike will probably not work as expected!

Often people use a filter string to display something like **.addr == 1.2.3.4** which will display all packets containing the IP address 1.2.3.4.

Then they use **.addr != 1.2.3.4** to see all packets not containing the IP address 1.2.3.4 in it. Unfortunately, this does **not** do the expected.

Instead, that expression will even be true for packets where either source or destination IP address equals 1.2.3.4. The reason for this, is that the expression **.addr != 1.2.3.4** must be read as "the packet contains a field named ip.addr with a value different from 1.2.3.4". As an IP datagram contains both a source and a destination address, the expression will evaluate to true whenever at least one of the two addresses differs from 1.2.3.4.

If you want to filter out all packets containing IP datagrams to or from IP address 1.2.3.4, then the correct filter is **!(.addr == 1.2.3.4)** as it reads "show me all the packets for which it is not true that a field named ip.addr exists with a value of 1.2.3.4", or in other words, "filter out all packets for which there are no occurrences of a field named ip.addr with the value 1.2.3.4".