**Wireshark Filters**

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

### 6.4.1. 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.

### 6.4.2. Comparing values

You can build display filters that compare values using a number of different comparison operators. They are shown in [Table 6.3, “Display Filter comparison operators”](http://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html#DispCompOps).

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

**Table 6.3. 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. [Table 6.4, “Display Filter Field Types”](http://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html#ChWorkFieldTypes) provides a list of the types and example of how to express them.

**Table 6.4. 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/" |

### 6.4.3. Combining expressions

You can combine filter expressions in Wireshark using the logical operators shown in [Table 6.5, “Display Filter Logical Operations”](http://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html#FiltLogOps)

**Table 6.5. 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. |

### 6.4.4. A common mistake

|  |  |
| --- | --- |
| [Warning] | **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 **ip.addr == 1.2.3.4** which will display all packets containing the IP address 1.2.3.4.

Then they use **ip.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 **ip.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.

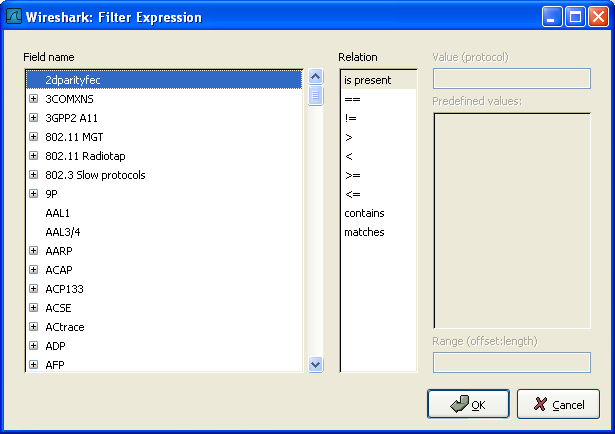
If you want to filter out all packets containing IP datagrams to or from IP address 1.2.3.4, then the correct filter is **!(ip.addr == 1.2.3.4)** as it reads "show me all the packets for which it is not true that a field named ip.addr exists with a value of 1.2.3.4", or in other words, "filter out all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

**6.5. The "Filter Expression" dialog box**

When you are accustomed to Wireshark's filtering system and know what labels you wish to use in your filters it can be very quick to simply type a filter string. However if you are new to Wireshark or are working with a slightly unfamiliar protocol it can be very confusing to try to figure out what to type. The Filter Expression dialog box helps with this.

|  |  |
| --- | --- |
| [Tip] | **Tip!** |
| The "Filter Expression" dialog box is an excellent way to learn how to write Wireshark display filter strings. |

**Figure 6.6. The "Filter Expression" dialog box**



When you first bring up the Filter Expression dialog box you are shown a tree list of field names, organized by protocol, and a box for selecting a relation.

**Field Name**

Select a protocol field from the protocol field tree. Every protocol with filterable fields is listed at the top level. (You can search for a particular protocol entry by entering the first few letters of the protocol name). By clicking on the "+" next to a protocol name you can get a list of the field names available for filtering for that protocol.

**Relation**

Select a relation from the list of available relation. The **is present** is a unary relation which is true if the selected field is present in a packet. All other listed relations are binary relations which require additional data (e.g. a **Value** to match) to complete.

When you select a field from the field name list and select a binary relation (such as the equality relation ==) you will be given the opportunity to enter a value, and possibly some range information.

**Value**

You may enter an appropriate value in the **Value** text box. The **Value** will also indicate the type of value for the **field name** you have selected (like character string).

**Predefined values**

Some of the protocol fields have predefined values available, much like enum's in C. If the selected protocol field has such values defined, you can choose one of them here.

**Range**

XXX - add an explanation here!

**OK**

When you have built a satisfactory expression click **OK** and a filter string will be built for you.

**Cancel**

You can leave the **Add Expression...** dialog box without any effect by clicking the **Cancel** button.

**Examples**

Show only [SMTP](http://wiki.wireshark.org/SMTP) (port 25) and [ICMP](http://wiki.wireshark.org/ICMP) traffic:

* tcp.port eq 25 or icmp

Show only traffic in the LAN (192.168.x.x), between workstations and servers -- no Internet:

* ip.src==192.168.0.0/16 and ip.dst==192.168.0.0/16

[TCP](http://wiki.wireshark.org/TCP) buffer full *-- Source is instructing Destination to stop sending data*

* tcp.window\_size == 0 && tcp.flags.reset != 1

Filter on Windows *-- Filter out noise, while watching Windows Client - DC exchanges*

* smb || nbns || dcerpc || nbss || dns

Sasser worm: *--What sasser really did--*

* ls\_ads.opnum==0x09

Match packets containing the (arbitrary) 3-byte sequence 0x81, 0x60, 0x03 at the beginning of the [UDP](http://wiki.wireshark.org/UDP) payload, skipping the 8-byte UDP header. Note that the values for the byte sequence implicitly are in hexadecimal only. *(Useful for matching homegrown packet protocols.)*

* udp[8:3]==81:60:03

The "slice" feature is also useful to filter on the vendor identifier part (OUI) of the MAC address, see the [Ethernet](http://wiki.wireshark.org/Ethernet) page for details. Thus you may restrict the display to only packets from a specific device manufacturer. E.g. for DELL machines only:

* eth.addr[0:3]==00:06:5B

It is also possible to search for characters appearing anywhere in a field or protocol by using the matches operator.

Match packets that contains the 3-byte sequence 0x81, 0x60, 0x03 anywhere in the UDP header or payload:

* udp contains 81:60:03

Match packets where SIP To-header contains the string "a1762" anywhere in the header:

* sip.To contains "a1762"

The matches operator makes it possible to search for text in string fields and byte sequences using a regular expression, using Perl regular expression syntax. Note: Wireshark needs to be built with libpcre in order to be able to use the matches operator.

Match HTTP requests where the last characters in the uri are the characters "gl=se":

* http.request.uri matches "gl=se$"

Note: The $ character is a PCRE punctuation character that matches the end of a string, in this case the end of http.request.uri field.

Filter by a protocol ( e.g. SIP ) and filter out unwanted IPs:

ip.src != xxx.xxx.xxx.xxx && ip.dst != xxx.xxx.xxx.xxx && sip

*[ Feel free to contribute more ]*

**Gotchas**

Some *filter fields* match against multiple *protocol fields*. For example, "ip.addr" matches against both the [IP](http://wiki.wireshark.org/IP) source and destination addresses in the IP header. The same is true for "tcp.port", "udp.port", "eth.addr", and others. It's important to note that

* ip.addr == 10.43.54.65

is equivalent to

ip.src == 10.43.54.65 or ip.dst == 10.43.54.65

This can be counterintuitive in some cases. Suppose we want to filter out any traffic to or from 10.43.54.65. We might try the following:

* ip.addr != 10.43.54.65

which is equivalent to

ip.src != 10.43.54.65 or ip.dst != 10.43.54.65

This translates to "pass all traffic except for traffic with a source IPv4 address of 10.43.54.65 **and** a destination IPv4 address of 10.43.54.65", which isn't what we wanted.

Instead we need to negate the expression, like so:

* ! ( ip.addr == 10.43.54.65 )

which is equivalent to

! (ip.src == 10.43.54.65 or ip.dst == 10.43.54.65)

This translates to "pass any traffic except with a source IPv4 address of 10.43.54.65 **or** a destination IPv4 address of 10.43.54.65", which is what we wanted.