Flightplan Visualizer (FV) Filter manual

In versions 1.14 of FV I implemented a filter combo-box on the main form which allows you to filter which flightplans are listed in the flightplan combo-box below. A tool-tip appears when you hoover the mouse over the filter combo-box and lists some examples of filters. Then in version 1.15 I added filters in the leg-search form, above the flightplan-, airline- and aircraft- list-views. Each of these still have a tool-tip showing many of the types on which you can filter, however many of the new filters added in version 1.15 can be setup to be very complex. For this reason I decided to make a small/separate manual explaining how these filters can be set up. This manual can be launched both from the Help menu (in the main form) and directly from the leg-search form.

Specific filters

In the first pages this manual will tell you about the generic usage of filters, that apply to all filters. Then the filters for each specific type (flightplan, airline, aircraft and airport) are described (they all share the same generic way of functioning, but have different element-types). If you want to look up a specific filters, then skip ahead to the sections listed below:

- **Flightplan** filter ⟵ *Use link to go directly to section*
- **Airline** filter ⟵ *Use link to go directly to section*
- **Aircraft** filter ⟵ *Use link to go directly to section*
- **Airport** filter ⟵ *Use link to go directly to section*

Filter validation

As you enter the filter-text it is constantly being validated. If the program detects an error in the filter-text it will change the text-color to **red** to indicate an error have been detected. As long as the filter-text is red the filter is inactive (the last correctly entered filter remains in use). When the text turns black it is not an indication that the filter is a "good filter". You can still have entered something that does not make sense. E.g. show all aircraft with more that 10 engines, or only show airlines that at the same time both are registered in USA and CHN (China) - as an airline can only be associated with a single country.

Keep it simple (stupid)

Even though the filters can be used to enter very complex filter-text, in its most basic form you can simply enter a short text/name and show all items containing this text in its name. E.g. in the flightplan filter you can enter `west` to only include flightplans with `west` in the name. Likewise you can enter `737` into an aircraft-filter to only show aircraft with `737` in their names (e.g. `Boeing 737`). The filter is case insensitive so you are free to use upper-/lower-case characters as you please. Once filtered you can select/deselect items and then enter another filter-text and again select/deselect as you pleases (rinse and repeat).

Wildcards

Most of the filter elements allows you to enter (DOS-like) wildcards. A wildcard is "place-holder" for none-, one- or multiple- characters. The asterisk (`*`) is the most commonly used wildcard character and it is at the same time a placeholder for none-, one- or multiple- characters. E.g. entering the flightplan filter text `ca*da` will include all flightplans with `canada` in their name, as the asterisk in this case is a placeholder for `na`. Don't forget that the asterisk can be a placeholder for many character split across many words so a (fictitious) flightplan called: `Balcan super-duper not danish airline` would also be included. However if you had a flightplan with `cada` in its name (e.g. `abracadabra`) it would be included also, as the asterisk can also be a placeholder for *nothing*.

Beside the asterisk the question-mark (`?`) is also a valid wildcard, however it is a wildcard for exactly one character. Entering the filter-text `ca?da` will not include airlines with `canada` in their name as `na` is 2 characters. Likewise `abracadabra` will not be included either as in this case there is no character between `ca` and `da`. However if you enter the filter-text as `ca??da` then flightplans with `canada` in their names would be include as `na` is eactly 2 characters long. Likewise if you enter the filter-text `bal?ic`, flightplans with `Baltic` in their name would be included, as the question-mark in this case is a placeholde for the `t`. The percent-sign (`%`) can be used in stead of the question-mark, and it functions in the exact same way (`?` is common in DOS wildcards, where `%` is common as a SQL wildcard). Last but not least the hash-mark (`#`) is a wildcard for a single character (like `?` and `%`), however the character have to be a digit (`0-9`). E.g. searching for aircraft using the filter-text `7#7` would include `717`, `727`, `737`, `747` … and so on, however if an aircraft was called `7M7` it would not be included as `M` is not a digit.

Automatic wildcards and suppression of these

To search for an airline with `west` in its name you should in theory have to enter the filter-text as `*west*`, meaning there can be *nothing* or *something* both before- and after "west". However to keep the filter as easy to use as possible, I will automatically add a prefixed (before) and suffixed (after) asterisk wildcard, so when you enter `west` I automatically change it into `*west*` inside the program. However there might be cases where you want to suppress these prefixed/suffixed wildcards from being added automatically. One example would be to include all flightplans that begins with `west` but exclude those that only have `west` somewhere else in their name (e.g. include `WestJet` but exclude `Southwest Airlines`). These automatic added wildcards can be suppressed using double-qoute (`"`) as the first- and/or last character in the filter-text. In case with finding all airlines that begin with `west` you would enter the filter text as `"west` (in this case the prefixed asterisk is suppressed – not added – however the suffixed asterisk is still added automatically).

Entering/separating multiple filter-elements (or'ed)

In the same filter-text you can enter multiple different filter-elements to search for, but doing so these need to be separated by comma (`,`). E.g. searching for all flightplans containing either `east`, `west`, `north` or `south` in their names you would enter the filter-text as `east,west,north,south`. You are allowed to insert spaces before/after the comma if you think it makes the text more readable, but it is not necessary/required, and they are simply ignored. As a software developer I would say that these elements are `or`'ed, meaning the result will contain all flightplans that contain `east` or `west` or `north` or `south` in their name.

In the next section we will look at different filter-types entered into the same filter-text. Filter sub-elements of the same type can be separated with a slash (`/`) or the pipe-char (`|`), whereas filter-elements of different types have to be separated by comma (`,`).

Filter-types

In the same filter-text you can enter multiple elements to search for. As stated above if the elements are of the same type you can (in most cases) separate them with a slash (`/`) or a pipe-char (`|`), but if the elements are of different types they need to be separated by comma (`,`). When you not explicitly specify the type, the search filter will have a **default type** that it uses. E.g. entering **`west`** into a flightplan filter, it will look for flightplans containing **`west`** in their name. When using the an airline/aircraft- filter the default type will both look in ICAO- and IATA-code, and the name of the airline/aircraft. The default filter for airports will only look in the ICAO-code of the airport (to be backward compatible). These are the most commonly elements to search for, hence that is why I chose them is as default, so you don't need to specify what you want to search for (again: no need to complicate things when not needed)

However if you want to search for other properties you will have to enter the name of type followed by an *equality-operator* (in most cases =) and then the value you want to search for. E.g. to find all summer flightplans you can enter the filter-text **`season=summer`**. If you at the same time want to limit the result to only include flightplans for a specific year you have to enter this as an additional element: **`season=summer, year=2018`**. As you see here we use comma to separate different types.

If you want to search for all flightplans made by the same author your can enter the filter-text as **`author=johan`** in order to find all flightplans made by Johan. If you want to find all flightplans made by either Johan or Kai you can enter the filter-text as **`author=johan, author=kai`**. However since these are both the same type (author) you can enter it in a way where you only specify the type a single time, but then add more values separated by slash (`/`), e.g. **`author=johan/kai`**. These are still regarded as two sub-elements of the same type (using the same equality-operator). Since they are the same type the result is or'ed as mentioned previously, so the result will include flightplans with **`johan`** or **`kai`** in the names of the authors.

If you at the same time want to search using different types (e.g. both default, and author) you can enter a filtertext of **`east,west,north,south,author=johan/kai`**. The elements of the same type are still or'ed. So whether the name contains **`east`**, **`west`**, **`north`** or **`south`** is not important (as long as it contains one of them), likewise it is not important if the flightplan is made by **`johan`** or **`kai`** (as long as it is one of them). However the 2 different sets of types are **and**'ed meaning that both type-criteria MUST be valid. E.g. Johan's "Danish Air Transport" is not included (as it does not contain "north", "east", "west", "south" in the name). Likewise "WestJet - Wi19-20" is not include as it is neither made by Johan or Kai (it is made by Allan Lee).

Forcing sub-elements of same type to be and'ed

As written before elements of the same types are or'ed (one of them must be true), whereas elements of different types are and'ed (each type MUST have at least one sub-element that is true). One example of when we would need to force elements of the same type to be and'ed, could be to include all flightplans that both contains **`air`** and **`cargo`** in their names. Simply writing both as separate elements would not work (either entered as **`air,cargo`** or **`air/cargo`**), as it would include all flightplans that had either of those words in their name (e.g. **`airBaltic`** that contains **`air`** even though it does not contain

cargo). Using a traditional wildcard we could chose to write **air*cargo**. This will include flightplans like **Polar Air Cargo**, but it would not include **Cargoair**. We can however fulfill this task by entering **air*cargo,cargo*air**. Since there are only 2 words, we are left with these 2 ways of combining them, but what if there had been 3 words we would want to search for, then we have to dig out our old math-knowledge of calculating the number of combinations.

We can however force all elements of a specific type to be and'ed in stead of being or'ed as they are by default. This is however not possible using the default filter type, so we have to use a named filter-type. The flightplan filter contains a name filter for the flightplan name simply called **name** so finding all flightplans with either "air" or "cargo" in its name can be written as: **name=air/cargo**. This does not solve our problem as we still include all fightplans with either **air** or **cargo** in their names, and we only want to include those that have both. To do this we have to tell the program that all elements of the name-type must be and'ed, and to do this you have to use the ampersand (**&**). This ampersand MUST be entered between the name of the type and the equality-sign (in this case =), hence the filter will be entered as: **name&=air/cargo**.

As soon as the ampersand is in use, ALL elements of that type are and'ed. So if you enter a filter-text as **name&=air,year=19,name=cargo**, the result will only contain flightplans that both have **air** and **cargo** in their names (and are from year 2019).

Equality operators

When you use the default filter-type (when you don't explicitly write the name of the filter-type) the eqaulity operator defaults to *equal* (=). E.g. if you simply enter **west** as a filter for flightplans, the included flightplans will be those with **west** in the name (hence the filter "have to be true"). However if you do specify the name of the filter you have the option of specifying another equality operator. So for instance you could set up a filter to exclude all flightplans with **west** in the name by writing **name!=west**, where **!=** is the way you write *not equal*. As an alternative (as used in Pascal/Delphi) you can write **<>** in stead of **!=** (it is just another supported way of writing *not equal*).

But you have to be careful if you want multiple sub-elements to be *not equal*. E.g. if you want all flightplans that don't have **east** nor **west** in their name, that you could be forgiven for thinking that it could be written as **name!=east/west**. However you have to remember that multiple sub-elements of the same type (in this case "name") are by default or'ed, meaning as long as one element is true, the other don't have to be. So in case with a flightplan called **WestJet** the first sub-element is true (**name!=east**) as this flightplan don't have **east** in the name. So in this case we have to use the ampersand to ensure all elements have to be true, so the filter should be written as **name&!=east/west**. Remember the ampersand MUST be written before the equality operator and *not equal* can be written as both **!=** or **<>**, however =! will not be allowed (nor will ><) and will change the text color to red to indicate an error.

All filter-types support both = and **!=** (inclusive **<>**) however only numeric sub-elements supports: **<** (less than), **<=** (less than or equal), **>** (greater than), and **>=** (greater than or equal). When using an aircraft filter you are able to specify the passenger count of the aircraft you want to include. E.g. you might write the filter-text as **passengercount>=20** to only include aircraft with 20 or more passengers (**=>** or **=<** are not valid and will turn the filter-text red to indicate an error).

But what if you want to both set a low- and a high limit (e.g. to include all aircraft with 20 to 50 passengers – both included). Again you have to use the ampersand. Writing the filter as **`passengercount>=20,passengercount<=50`** will not work, as the sub-elements are or'ed and ALL aircraft will either have a passengercount that is 20 or greater, or 50 and less (hence ALL aircraft will be included). Many of the filter-types are *overloaded* so there are often multiple ways to write the same filter-type, so **`passengercount`** can also be written simply as **`pax`**. So the correct way to write this filter is: **`pax&>=20,pax<=50`**. As stated before as soon as the filter type has been written once with the ampersand, ALL instances of that filter-type will be and'ed so the same filter can also be written as **pax>=20, pax&<=50**.

The equality operator have to come after the name of the filter (an ampersand between them are allowed), so it is not allowed to write the filter as **`pax&>=20/<=50`**. It is perfectly allowed to write **`pax&>=20/50`**, but it makes no sense to do so. Since we will only include aircraft fulfilling both sub-elements hence only aircraft with 50 or more passengers will be included. Also it is allowed to write the filter as **`pax&=20/50`** however this makes even less sense. In this case you specify you only want to include all aircraft that both have a passenger count of 20 and a passenger count of 50. However as you use the ampersand both have to be true at the same time, and its impossible for the same aircraft-type to both have a passenger count of 20 and of 50, hence NO aircraft will be included. The filter is not invalid and as such will not turn the text red. However it would be just as stupid as trying to list all aircraft suitable for more than 10000 passengers. Had you in stead entered the filter without ampersand (e.g. **`pax=20/50`**) the elements are or'ed meaning the result would include all aircraft that either had an exact passenger count of 20 or 50.

Don't complicate things

Always choose the most lazy approach to write the filter. In an aircraft filter you can specify the engine-count (number of engines) using the short-hand **`ec`**. So to find all aircraft with 1 or 2 engines you can write it is either: **`ec=1/2`** or **`ec<3`**. In theory these filters are not the same as the last would also include aircraft with 0 engines. However at least as of now there are not such aircraft, hence the result-sets are the same. Likewise don't over complicate the filter. If you only want to select a few specific 737 types like the max'es you could write the filter as **`name&=737/max`** in order to only show those aircraft that both contain **`737`** and **`max`** in their name (both). But being lazy I would simply write **`max`** as I know there are fewer aircraft with **`max`** in their name as there are with **`737`**. Likewise to only select the 747 freighters I would probably only search for **`747`**. This will show all 747's (including the pax and mixed), but it is still only a few aircraft that will be shown, so I can quickly put a check-mark next to the ones I need (it is still a lot quicker than writing a complex filter).

Multiple ways to write the same filter-type

As stated previously the same filter-text can normally be written in multiple ways, so I suggest you memorize the short-hand versions of each type as they are faster to type. However the longer (full) versions of the type names might be easier to remember. Since there are many versions of each filter-type that you are allowed to enter, you are in luck as long as you remember just one of these. E.g. the passenger count we looked at before can both be written as "passengercount", "paxcount", "passengers", "passenger", "pax" or simply as “pc”.

What I have written here on the first pages are generic for all filters and describes the general behavior of all the filters. In the next pages I will instead describe each different filter, listing the various filter-types that can be used for each filter.

Flightplan filter

The **default** filter (used when not specifying a filter-type name) for flightplan filters is the **name of the flightplan**. Remember the (default) equality-operator for the default type is always = (equal), and all elements are or'ed, as you are not able to specify neither an equality operator nor an ampersand.

Name (can contain wildcards)

You are able to explicitly specifying the name filter-type (e.g. `name=west`). By doing so you are able to use the not-equal equality-operator and the ampersand if you need to.
Alternatives: `name|nm|n`
Example: **`name=west`**
Example: **`n&=air/cargo`**

Author (can contain wildcards)

By using the author filter-type you are able to specify a specific author, in order to only list flightplans made by that author.
Alternative: `author|athr|ath|a`
Example: **`author=johan/kai`**

Provider (can contain wildcards)

Johan Clausen both releases flightplans using AIG as the provider or using this own provider name JCAI. If you simply use the author type you can find all flightplans made by Johan, but if you only want to see those he released with JCAI as the provider you have to use the provider type.
Alternative: `provider|prvdr|prvd|prv|pvd|p`
Example: **`provider=jcai`**

Year (numeric value, can't contain wildcards)

Using the year filter-type you are able to specify which year(s) to include. In this aspect it is important to remember that winter flightplans are normally containing 2 different years. E.g. "Wi19-20" meaning the winter of (late) 2019 and (early) 2020. However it is the first year that is, used so in this case you would have to use **`year=2019`** (combined with **`season=winter`** if you only want to exclude summer 2019). The year can be written either as a four digit number (e.g. `2019`) or a two digit number (e.g. `19`). When entered with 2 digits `20` will be prefixed (e.g. turning `19` into `2019`). This value is only prefixed behind the scene, hence the entered filter-text will not change. Since Year is a numeric value you are also able to use the equality-operators **<**, **<=**, **>**, and **>=**.
Alternative: `year|yr|y`
Example: **`year=2019,season=winter`**
Example: **`year<=2019`**
Example: **`yr&>=2016,yr<=2018`**
Example: **`y=16/17/18`**

Season (fixed values – without wildcards)

You are not able to use a wildcard with this filter-type, but in stead you have to use either **`summer`** or **`winter`**. Both are however available in shorter alternative versions. You will most likely always use this filter-type together with the year-type.

Alternative: `season|ssn|sn|s`
Alternative(summer): `summer|smr|su|s|1`
Alternative(winter): `winter|wntr|wi|w|2`
Example: **`year=2019,season=winter`**
Example: **`y=19,s=wi`**

Yearseason (numeric value, can't contain wildcards)

Yearseason is a combination of year and season into the same filter. Not only can it be used as short-hand of filtering on both year and season, but it also allows you to perform operations that are not possible with the two separate filter-types. With year and season you can not setup a combined filter to include all flights (from all years) before winter 2019. When using the Yearseason filter-type you have to use a 3 or 5 digit number, where the first 2 or 4 are the year, and the last is the season (**`1=`**summer,**`2`**=winter). However you are also allowed to enter the seasons as **`S`** (Summer) or **`W`** (winter). In case you do so, these are simply converted into the digits `1` or `2` in the software (however not displayed as such).
Alternative: `yearseason|yrssn|yrsn|yrs|yssn|ysn|ys`
Example: **`yearseason<2019W`**
Example: **`ys&>=20151,ys<20182`**

Folder (can contain wildcards)

If you go into the "Enable/Disable flightplans" form you can setup various sub-folders at move your flightplans into these folders. Using the folder filter-type you can set up a filter to only include flightplans from certain folders (these still needs to be set as active in "Enable/Disable flightplans". I have all my BizJet flightplans in a sub-folder called BizJet which I can include or exclude using this filter. The root-folder is called **`ROOT`** (but not case sensitive).
Alternative: `folder|fldr|fld|fdr|f`
Example: **`folder=bizjet`**
Example: **`folder!=ROOT`**

Airline filter

The **default** filter (used when not specifying a filter-type name) for airline filters is a combination of **ICAO**-/**IATA**-code and the **name** of the airline (as long as either of the three contains the text you entered, that airline will be included). Remember the (default) equality-operator for the default type is always = (equal), and all elements are or'ed, as you are not able to specify neither an equality operator nor an ampersand.

Name (can contain wildcards)

You are able to explicitly specifying the name filter-type (e.g. `name=west`). By doing so you are able to use the not-equal equality-operator and the ampersand if you need to.
Alternatives: `name|nm|n`
Example: **`name=west`**
Example: **`n&=air/cargo`**

ICAO (can contain wildcards)

The icao-type lets you search for airlines with a specific ICAO-code.
Alternatives: `icao|ic`
Example: **`icao=sas`**
Example: **`ic=klm/afr/dhl`**

IATA (can contain wildcards)

The iata-type lets you search for airlines with a specific IATA-code.
Alternatives: `iata|ia`
Example: **`iata=sk`**
Example: **`ia=kl/af/lh`**

Country (can contain wildcards)

Using the country-type you are able to search for airlines registered in a certain country. However an airline can only be registered in a single country. E.g. SAS (Scandinavian Airlines) is considered both a Danish-, Norwegian- and Swedish airline, however only registered as Swedish. Searching for airlines registered in certain countries, you have to use the 3-letter ISO country codes (e.g. `DNK` for Denmark, `NOR` for Norway and `SWE` for Sweden). If you drop-down the country combo-box in the right side of the leg-search form you can see the ISO3 code for each country in use.
Alternatives: `country|ctry|ctr|ct`
Example: **`country=usa`**
Example: **`ct=dnk/nor/swe/fin/isl`**

Region (can contain wildcards)

Using the region-type you are able to search for airlines located in certain regions (based on the country in where they are registered). Searching for airlines located in certain regions, you have to use the 3-letter region codes (e.g. `EUR` for Europe). If you drop-down the region combo-box you can see the 3-letter code for each region.
Alternatives: `region|regn|reg|rgn|rg`
Example: **`region=eur`**
Example: **`rg=asi/oce`**

Subregion (can contain wildcards)

Using the subregion-type you are able to search for airlines located in certain subregions (based on the country in where they are registered). Searching for airlines located in certain subregions, you have to use the 3-letter subregion codes (e.g. `EUR` for Europe). If you drop-down the subregion combo-box you can see the 3-letter code for each subregion.
Alternatives: `subregion|subreg|subrg|sbrg|sreg|srg|sr`
Example: **`subregion=naf`**
Example: **`sr=sas/sea/eas`**
Example: **`sr&!=pol/mel/mic`**

Callsign (can contain wildcards)

The callsign-type lets you search for airlines with a specific callsign. You can see the callsign in the airline browser/search form. However if you have to look up an aircraft to find its callsign, it kind of defeats the purpose. In that case either filter by ICAO, IATA or name in stead.
Alternatives: `calsign|call|cs`
Example: **`callsign=scandinavian`**
Example: **`cs=hansaline/brickyard`**

Aircraft filter

The **default** filter (used when not specifying a filter-type name) for aircraft filters is a combination of **ICAO**-/**IATA**-code and the **name** of the aircraft (as long as either of the three contains the text you entered, that aircraft will be included). Remember the (default) equality-operator for the default type is always = (equal), and all elements are or'ed, as you are not able to specify neither an equality operator nor an ampersand.

Name (can contain wildcards)

You are able to explicitly specifying the name filter-type (e.g. **`name=boeing`**). By doing so you are able to use the not-equal equality-operator and the ampersand if you need to.
Alternatives: `name|nm|n`
Example: **`name=boeing`**
Example: **`n&=airbus/320`**

ICAO (can contain wildcards)

The icao-type lets you search for aircraft with a specific ICAO-code.
Alternatives: `icao|ic`
Example: **`icao=B772`**
Example: **`ic=A332/A333`**

IATA (can contain wildcards)

The iata-type lets you search for aircraft with a specific IATA-code.
Alternatives: `iata|ia`
Example: **`iata=DH4`**
Example: **`ia=74Y/74N`**

Enginetype (fixed values – without wildcards)

For each aircraft I have specified its engine type as either **`jet`**, **`turboprop`** or **`prop`**. Using the **`enginetype`** filter-type you are able to filter to only show aircraft with the filtered enginetype.
Alternatives: `enginetype|engtyp|engtp|et`
Alternatives(jet): `jet|jt|j`
Alternatives(turboprop): `turboprop|turbo|tp`
Alternatives(prop): propeller|prop|p
Example: **`enginetype=jet`**
Example: **`et!=prop`**

Enginecount (numeric value, can't contain wildcards)

Beside filtering on the type of engines you can also filter on the number of engines by using the **`enginecount`** filter-type.
Alternatives: `enginecount|engcount|engcnt|ec`
Example: **`enginecount=2`**
Example: **`ec>=3`**

Usage (fixed values – without wildcards)

Using the usage filter-type you can filter on how the aircraft are used (e.g. `passenger`, `freight`, or various types of military purpose). This usage is not based on the flightplans, as the flightplans don't contain such info. In stead its based on the aircraft types (e.g. an Airbus 320 is obviously a commercial passenger aircraft). *Flightplan visualiser don't come bundled with any military flightplans, so you have to import these yourself.* Beside pax/cargo, there is also a `mixed/combi` usage (e.g. the 747 comes in a mixed/combi version). When an aircraft comes in a pax- and a freight-version these are normally added as two separate aircraft with their unique iata/icao-codes (e.g. "77L, Boeing 777-200LR Pax" and "77X, Boeing 777 Freighter"). However there are aircraft-types used both for passenger and freight, where there are no two sets of unique iata/icao-codes. In these cases the datafile will only contain a single aircraft and the usage of this aircraft is specified as `paxfreight`. Also there are three major values called `civilian` (all non-military usage both pax and freight), commercial (all civilian except GA) and `military` (all non-civilian usage both pax and freight).
Alternatives: `usage|use|us|u`
Alternatives(passenger): `passengers|passenger|pax`
Alternatives(freight): `freighter|freight|cargo`
Alternatives(paxfreight): `paxfreighter|paxfreight|freighterpax|freightpax|paxcargo|cargopax`
Alternatives(combi): `combi|combo|mixed`
Alternatives(ga): `generalaviation|general|ga`
Alternatives(utility): `utility|util|ut`
Alternatives(civilian): `civilian|civil|civ`
Alternatives(commercial): `commercial|com`
Alternatives(military): `military|mil`
Alternatives(milcombat): `milcombat|combat`
Alternatives(milfreight): `milfreighter|milfreight|milcargo`
Alternatives(milutility): `milutility|milutil|milut`
Example: **`usage=freight/paxfreight/combi`**
Example: **`usage=cargo/milcargo`**
Example: **`usage=passenger,enginecount=2,enginetype=turboprop`**

Passengercount (numeric value, can't contain wildcards)

For each passenger aircraft-type I have registered a maximum passengercount (this value is 0 for freight aircraft). When an airline purchase an aircraft they can choose between different seating-configurations, so the value I have chosen might vary from the passengercount from a specific configuration of set aircraft-type (e.g. it might differ from a 3rd party aircraft model you are using in your sim).
Alternatives: `passengercount|passengers|passenger|paxcount|pax|pc`
Example: **`passengercount>300`**
Example: **`pax&>=20,pax<50`**

Favorite (fixed values – without wildcards)

Using simple fixed values you are able to filter aircraft marked as favorites.
Alternatives: `favorites|favorite|fav|fv`
Alternatives(true): `true|yes|t|y`
Alternatives(false): `false|no|f|n`

Example: **`favorites=true`**
Example: **`fav=yes,usage=cargo`**

Typerate (fixed values – without wildcards)

I have categorized the aircraft into multiple typerate categories where the aircraft have been categorized according to enginetype and size (based on `EOW/MTOW`). Please remark the "jet" categories contains a "minor" classification that the "prop" and "turboprop" categories does not (simply to have an extra category in this large segment of aircraft).
Alternatives: `typerating|typerate|type|rate|tr`
Alternatives(single engine prop): `PropSingle`
Alternatives(small multi engine prop): `PropMultiSmall`
Alternatives(medium multi engine prop): `PropMultiMedium`
Alternatives(large multi engine prop): `PropMultiLarge`
Alternatives(huge multi engine prop): `PropMultiHuge`
Alternatives(single engine turboprop): `TurbopropSingle|TurboSingle`
Alternatives(small multi engine turboprop): `TurbopropMultiSmall|TurboMultiSmall`
Alternatives(medium multi engine turboprop): `TurbopropMultiMedium|TurboMultiMedium`
Alternatives(large multi engine turboprop): `TurbopropMultiLarge|TurboMultiLarge`
Alternatives(huge multi engine turboprop): `TurbopropMultiHuge|TurboMultiHugh`
Alternatives(business jet): `BusinessJet|BizzJet|BizJet`
Alternatives(small jet): `JetSmall`
Alternatives(minor jet): `JetMinor`
Alternatives(medium jet): `JetMedium`
Alternatives(huge jet): `JetHuge`
Example: **`typerate=bizzjet|jetsmall`**
Example: **`tr=PropSingle|TurboSingle`**

Size (fixed values – without wildcards)

While the (long) names used in the typerate filter-type can be hard to remember they do offer better flexibility selecting various sizes of aircraft across multiple engine-types. However when you don't need this flexibillity, you can simply use the Size filter-type. But for the best result it should be used together with the enginetype, as the sizes are not comparable between between different engine-types:
Alternatives: `size|sz`
Alternatives(single engine): `single|sgl`
Alternatives(small multi engine): `small|sml`
Alternatives(minor multi engine – only jets): `minor|min|mnr`
Alternatives(medium multi engine): `medium|med`
Alternatives(large multi engine): `large|lrg|lar`
Alternatives(huge multi engine): `huge|hge|hug|hg`
Examples: **`enginetype=jet,size=huge`**
Examples: **`enginetype=turboprop,size=small/medium`**

Empty operating weight (numeric value, can't contain wildcards)

In case you disagree with my size categorization (using the `typerate`- or `size` types) you can filter by `emptyoperatingweight` (or simply `EOW`). This is a numeric value hence you can use all of the supported equality-operators, and the weight is specified in lbs.

Alternatives: `emptyoperationweight|emptyweight|eow|ew`
Example: **`eow<=10000`**
Example: **`eow&>=50000,eow<70000`**

Maximum take-off weight (numeric value, can't contain wildcards)

Like with `EOW` you can use `maximumtakeoffweight` to do your own size categorization using the `MTOW` filter-type. This is a numeric value hence you can use all of the supported equality-operators, and the weight is specified in lbs.
Alternatives: `maximumtakeoffweight|maxtakeoffweight|takeoffweight|mtow|tow|tw`
Example: **`mtow&>=300000,mtow<400000`**

Range (numeric value, can't contain wildcards)

You can filter by the range (the distance between airports that can be served by the filtered aircraft). In real-life the range (distance) an aircraft can cover is based very much on the payload (how many passengers/how much cargo) it carries. This is however not possible to take that into consideration here. So the range for each aircraft is a static value, and as such it might differ from other data you might have at hand (e.g. documentation for a 3rd party model).
Alternatives: `range|rng|rge`
Example: **`range<1000`**
Example: **`rng>=5000`**
Example: **`rng&>=1000,rng<3000`**

Airport filter

The **default** filter (used when not specifying a filter-type name) for airport filters is the **ICAO**-code of the airport (to keep working as the old text-entry prior to version 1.15). Remember the (default) equality-operator for the default type is always = (equal), and all elements are or'ed, as you are not able to specify neither an equality operator nor an ampersand.

ICAO (can contain wildcards)

The icao-type lets you search for airports with a specific ICAO-code. By doing so you are able to use the not-equal equality-operator and the ampersand if you need to.
Alternatives: `icao|ic`
Example: **`icao=kbos`**
Example: **`ic=ekch/engm/essa`**
Example: **`ic&!=ekch/ekbi/ekyt,ic=ek??`**

IATA (can contain wildcards)

The iata-type lets you search for airports with a specific IATA-code.
Alternatives: `iata|ia`
Example: **`iata=bos`**
Example: **`ia=chp/bll/aal`**

Name (can contain wildcards)

You are able to explicitly specifying the name filter-type containing the name of the airport (e.g. **`name=kastrup`**). The name is extracted from the scenery by (Pete Dowson's) MakeRunways, so the name might have changed since the scenery was released/updated. *(remember you should always run MakeRunways and copy the files mentioned in the manual, after having installed/updated your scenery, to have the latest data being available for FV).*
Alternatives: `name|nm|n`
Example: **`name=kastrup/gardemoen/arlanda`**
Example: **`n&=stevens/intl`**

City (can contain wildcards)

Using the city-type you are able to search for airports located in certain cities. In case you are filtering by a city-name existing in multiple countries, you might need to combine the city-filter with the country-filter.
Alternatives: `city`
Example: **`city=boston`**
Example: **`city=copenhagen/oslo/stockholm`**

State (can contain wildcards)

Using the state-type you are able to search for airports located in certain states. If you are in doubt as how the states are named, you can hover the mouse over the airport markers on the main map, and read the name of the state in the tool-tip window (the name of the state – if applicable – is listed in parentheses after the country-name). *You might need to enable some of the marker check-boxes in the lower/left-corner of the main-form in order to see the markers you might be looking for.*

Alternatives: `state|st`
Example: **`state=illinois`**
Example: **`st=new south wales/queensland`**

Country (can contain wildcards)
Using the country-type you are able to search for airports located in certain countries. If you don’t need to specify multiple countries or other filter-types, it will be quicker to simply use the country combo-box to pick that country. Searching for airports located in certain countries, you have to use the 3-letter ISO country codes (e.g. `DNK` for Denmark, `NOR` for Norway and `SWE` for Sweden). If you drop-down the country combo-box you can see the ISO3 code for each country in use.
Alternatives: `country|ctry|ctr|ct`
Example: **`country=usa`**
Example: **`ct=dnk/nor/swe/fin/isl`**

Region (can contain wildcards)
Using the region-type you are able to search for airports located in certain regions. If you don’t need to specify multiple regions or other filter-types, it will be quicker to simply use the region combo-box to pick that region. Searching for airports located in certain regions, you have to use the 3-letter region codes (e.g. `EUR` for Europe). If you drop-down the region combo-box you can see the 3-letter code for each region.
Alternatives: `region|regn|reg|rgn|rg`
Example: **`region=eur`**
Example: **`rg=asi/oce`**

Subregion (can contain wildcards)
Using the subregion-type you are able to search for airports located in certain subregions. If you don’t need to specify multiple subregions or other filter-types, it will be quicker to simply use the subregion combo-box to pick that region. Searching for airports located in certain subregions, you have to use the 3-letter subregion codes (e.g. `EUR` for Europe). If you drop-down the subregion combo-box you can see the 3-letter code for each subregion.
Alternatives: `subregion|subreg|subrg|sbrg|sreg|srg|sr`
Example: **`subregion=naf`**
Example: **`sr=sas/sea/eas`**
Example: **`sr&!=pol/mel/mic`**