Filters in Aladin

Thomas Boch

September 19th, 2006

Version 1.16

1 Introduction - What can I do with these filters

Filters are a new feature in Aladin allowing one to customize the display of catalogue planes in Aladin.

Basically (without filters), a catalogue plane :

- uses only 2 parameters (position : RA and DEC) for displaying sources
- displays the same symbol and the same color for each catalogue object, at the corresponding sky position

Thanks to the filters feature, it is possible to :

- combine several catalogue parameters with arithmetic operators ($+ - * / \hat{}$)
- set constraints on parameters (or combinations of parameters) to perform selections
- visualize any parameter by changing the symbols' shape / color / size / text. All these attributes can be customized as functions of catalogue parameters (or combination of parameters)

Notice that the position of visible symbols will of course remain their sky position.

Briefly, filters allow you not only to *select* sources on the basis of constraints, but also to *customize* the way a *catalog plane* is displayed, using the catalogue *contents*.

Figure 1 is an example of what you can do with filters : we have here splitted sources into 3 classes according to the value of their photographic B magnitude.

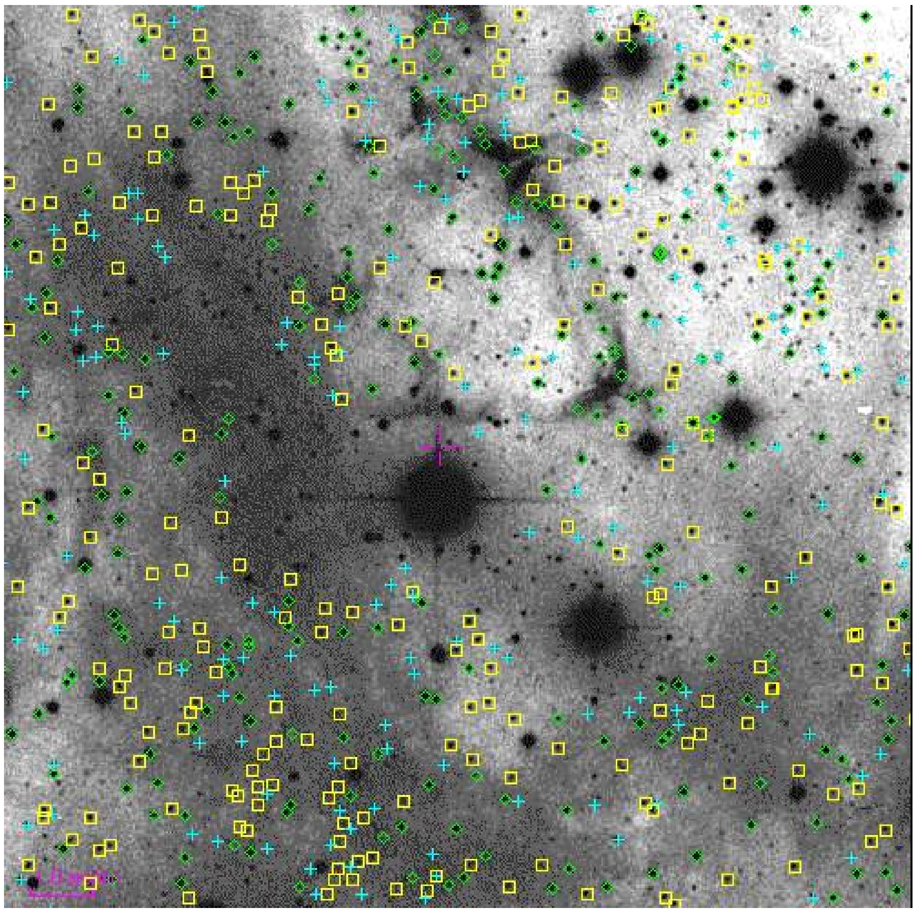


Figure 1: Example of filter result

2 Syntax

A **Filter** consists of a list of **Constraints** and associated **Actions**.
Example of filter :

```
Constraint 1 { Action 1
              Action 2 }
Constraint 2 { Action 3 }
```

A **Constraint** is a set of **Conditions** combined by logical operators (AND, OR).
Example of constraint : `( Condition 1 && Condition 2 ) || Condition 3`

A **Condition** applies to **Parameters** (or combination of parameters).
Examples of conditions :

```
Parameter >= Value
Parameter 1 - Parameter 2 = Value
```

Parameters are specified via the corresponding UCDs or column names.
Examples of parameters :
via UCD : `$[PHOT_JHN_V]`
via column name : `${Vmag}`

A **Value** can be a numeric value or a string value.

The global structure is quite similar to a *switch* structure in C which would have a *break* statement at the end of each *case*. It means that if a source verifies one constraint, the program won't run into following cases for this source.

Note : for a given constraint, the *list of actions* immediately follows the constraint and must be *embraced by brackets* `{}`. Each *action* has to be *separated* either by a **carriage return** or by a **semicolon**.

Figure 2 illustrates how a filter is applied to a set of sources. If a source verifies the first constraint, then each corresponding action is made and the next source is processed. If it doesn't, we check whether the source verifies the next constraint, and so on until there are no more blocks. If the source verifies no constraint at all, it is not drawn.

2.1 Pointing out columns

In constraints and in the parameters of actions, UCDs and column names have to be pointed out in a specific way.

The syntax is :

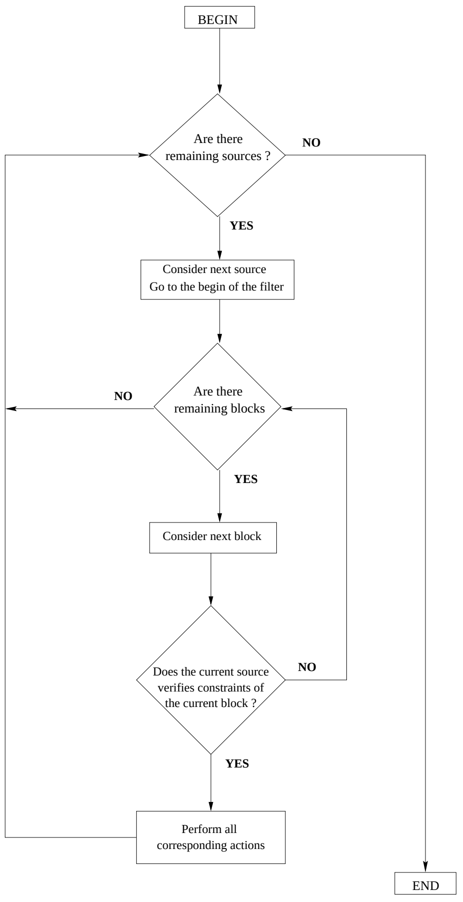

BEGIN
Are there remaining sources ?
NO
YES
Consider next source
Go to the begin of the filter
Are there remaining blocks
NO
YES
Consider next block
Does the current source verifies constraints of the current block ?
NO
YES
Perform all corresponding actions
END

Figure 2: How a filter processes a set of sources

- Parameters always begin with `$` (dollar sign)
- UCDs are enclosed in square brackets `[]`. E.g: `$[POS_EQ_PMDEC]` or `$[PHOT_FLUX_RATIO]` are valid UCD names.
- Column names are enclosed in braces `{}`. `${LumRat}` or `${Imag}` are valid column names.

UCD names are case insensitive (in fact, they are automatically converted to uppercase) whereas *column names* are **case sensitive**.

The star `*` and the question mark `?` can be used as wildcards in UCDs or column names. For instance, `$[PHOT*]` will point out the first column name being tagged by a UCD beginning by `PHOT`.

2.2 Comments

You can put comments into your filters. Each line beginning with "#" will be ignored.
E.g :

```
# This is a comment
```

2.3 Syntax of arithmetic expressions using UCDs/columns

You can use the following operators : +,-,*,/,^ to combine catalogue parameters. These parameters are specified via the corresponding UCDs or column names. They can be used to define constraints but also in some action parameters.
E.g:

```
${Bjmag}-$[PHOT_JHN_V]>1 {
# Do some action
...
draw ellipse( $[EXTENSION_RAD],
              $[EXTENSION_RAD]*(1-$[PHYS_ECCENTRICITY]^2)^0.5,
              $[POS_POS-ANG] )
}
```

Moreover, you can use the following functions:

- *abs* : absolute value
- *cos* : cosinus
- *deg2rad* : convert degrees to radians
- *exp* : exponential
- *ln* : natural logarithm (base e)
- *log* : base 10 logarithm
- *rad2deg* : convert radians to degrees
- *sin* : sinus
- *sqrt* : square root
- *tan* : tangent

E.g:

```
log(abs(${Fi})/${Fx})>44 {draw}
```

2.4 Syntax of Conditions

A **Condition** is an arithmetic combination of **Parameters** followed by a **Comparison Operator** followed by a **Value**.

The table below summarizes the allowed comparison operators for a numeric value.

Operator	**Meaning**	**Example**
$=$	Equality	$= 1$
$!=$	Inequality	$!= 1$
$>=$	Greater or equals	$>= 12.0$
$>$	Strictly greater	> 12.0
$<=$	Less or equals	$<= 12.0$
$<$	Strictly less	< 12.0

The table below summarizes the allowed comparison operators for a string value.

Operator	**Meaning**	**Example**
$=$	String equality (case sensitive)	$=$ "*Galaxy*"
$!=$	String inequality (case sensitive)	$!=$ "*UV*"

When performing a comparison on a string, you can only use one parameter, *ie* combination of parameters is not allowed in this case.
It is allowed to use the star `*` and the question mark `?` as wildcards when performing string comparison.
E.g:

```
# Draw in green sources whose name start with "IRAS"
$[ID_MAIN]="IRAS*" {draw green}
# Draw in red sources with object type "GNe" or "BNe" or "DNe" ...
${otyp}="?Ne" {draw red}
```

Remark: in order to escape the wildcard star `*` character, prefix it with a backslash.

Note : there is a special condition called *undefined(ucd or column name)* which is true if the entered ucd/column name is not present for the current object.
E.g:

```
# Process first sources without the column Bjmag --> draw them black
undefined(${Bjmag}) {
    draw black
}
# Process sources with column Bjmag
...
```

2.5 Syntax of Constraints

The logical operators used to combine conditions are **&&** – logical AND – and || – logical OR.
E.g:

```
($[PHOT_PHG_B]<16 && $[PHOT_PHG_R]<15) || $[CLASS_OBJECT]="Star" {
    draw
}
```

A constraint can also be empty. In such a case, the corresponding actions will apply to all sources.
E.g:

```
# No constraint, the action block is applied to all sources
{draw blue}
```

2.6 Syntax of Actions

Actions allow to change the appearance of symbols in a catalogue plane.
Available actions are :

- *hide*, which does nothing. This action is useful to hide a part of the plane sources.
- *draw ....* When used without parameters, this action just draws the source as usual using the default shape and color of the plane. It also has 2 optional parameters which allow to customize both *shape* and *color* (explained afterwards). **E.g:** draw green square
- You can also draw a *string*, being either a constant string or the content of a column. The color can also be specified. **E.g:** `draw $[CLASS_OBJECT] blue`

2.6.1 How to define a color

You can specify the color you want to assign to either a shape or a string. The color function is optional and can take place be either after the *draw* keyword, or after the optional shape function. E.g:

```
{
# Color function after the "draw" keyword
draw blue square
# Color function after the shape function
draw circle(-$[PHOT_PHG_B]) #00ff00
}
```

There are different ways to define a color :

- #rrggbb, where rr, gg, bb are respectively the values of red, green and blue components of the color in hexadecimal. **E.g:** `{draw #44dd99}`
- You can enter predefined color names. Allowed values are black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red white, yellow. **E.g:** `{draw red}`
- *rgb* function allows to define a color whose components depend on some column values. For each component, the max and the min values are computed, so that each value is normalized in order to be in the range 0-255. **E.g:** `{draw rgb($[PHOT_PHG_B],0,-$[PHOT_PHG_B])}`
- *rainbow* function has a single parameter in the range 0-1. This filter aims at going through the whole visible spectrum (blue-violet to red, 0 to 1). If the parameter is variable (for instance a column name), the values are normalized between 0 and 1. This can be useful if you want to visualize a color index. **E.g:** `{draw rainbow($[PHOT_JHN_U-B])}`

2.6.2 How do I customize the shape

You can customize the shape of sources. It is an optional parameter, which takes place either just after the *draw* keyword, or after the color function.

There are shapes without parameter : they are the same as those in the properties of a catalog plane :

* square
* rhomb
* cross
* plus
* dot
* microdot

E.g:

```
# Filter drawing different shapes according to the class of the object

# Draw a plus for Star objects
$[CLASS_OBJECT]="Star" {draw plus}
# Draw a rhomb for Radio objects
$[CLASS_OBJECT]="Radio" {draw rhomb}
# Draw a dot for other objects
{draw dot}
```

Furthermore, there are some shape functions requiring parameters :

* the *circle* function is used to draw a circle whose radius depends on a parameter which can be any column or combination of columns. The typical use of this function is drawing circles according to magnitude values. There are 2 optional parameters to define the minimum and the maximum *radius* size **in pixels** (otherwise, these values are set by default). Parameter values are normalized to fit into the range.
  The size of each circle is set as a number of pixels *relative to the underlying reference image*, which means that *circles scale up* when the user zooms in.
  E.g:

  ```
  # Draw circles according to Johnson B magnitude
  {draw circle(-$[PHOT_JHN_B])}

  # Draw circles according to Johnson B magnitude
  # Set min value to 3, max value to 40
  {draw circle(-$[PHOT_JHN_B],3,40)}
  ```

* the *fillcircle* function is similar to the *circle* one. It draws a filled circle instead of an empty one.

* the *fixedcircle* function is similar to the *fillcircle* one, except that the size of each circle (as a number of pixels) is relative to the user's screen, meaning that *fixedcircles do not scale up.*

* the *ellipse* function is typically used to draw dimension ellipses or error ellipses. It takes 3 parameters : the semi-major axis, the semi-minor axis. If the unit of one parameter is missing or not understood, semi-major and semi-minor axis are assumed to be in arcsec, and the position angle is assumed to be in degrees. If the position angle can't be retrieved or is empty, its value is assumed to be zero.
  E.g:

  ```
  # This filter draws dimension ellipses for Simbad sources
  # and GSC2 sources
  {
  # Ellipses for Simbad
  draw ellipse( 0.5*${DimMa}, 0.5*${DimMi}, ${DimPA} )
  # Ellipses for GSC2
  draw ellipse( $[EXTENSION_RAD],
                $[EXTENSION_RAD]*(1-$[PHYS_ECCENTRICITY]^2)^0.5,
                $[POS_POS-ANG] )
  }
  ```

* the *pm* function was designed to allow the visualization of proper motion of stars. It takes 2 parameters which are the proper motion in right ascension and the proper motion in declination, and draws an array corresponding to this proper motion. If units are missing or could not be understood, both parameters are assumed to be in mas/yr. By default, a proper motion of 1mas/yr is displayed by an array of 1arcsec.
  E.g:

  ```
  # Draws proper motions
  # 1mas/yr will correspond to an array of 5arcsec
  {draw pm(5*$[POS_EQ_PMRA],5*$[POS_EQ_PMDEC])}
  ```

* use the *rectangle* function to draw a rectangle by specifying its width, height, and position angle. If the unit of one parameter is missing or not understood, width and height are assumed to be in arcsec, and the position angle is assumed to be in degrees. If the position angle can't be retrieved or is empty, its value is set to zero.
  E.g:

  ```
  {
  # Draw a rectangle with dimensions and orientation
  # according to given field values in the catalogue
  draw rectangle( ${width}, ${height}, ${posAngle} )
  }
  ```

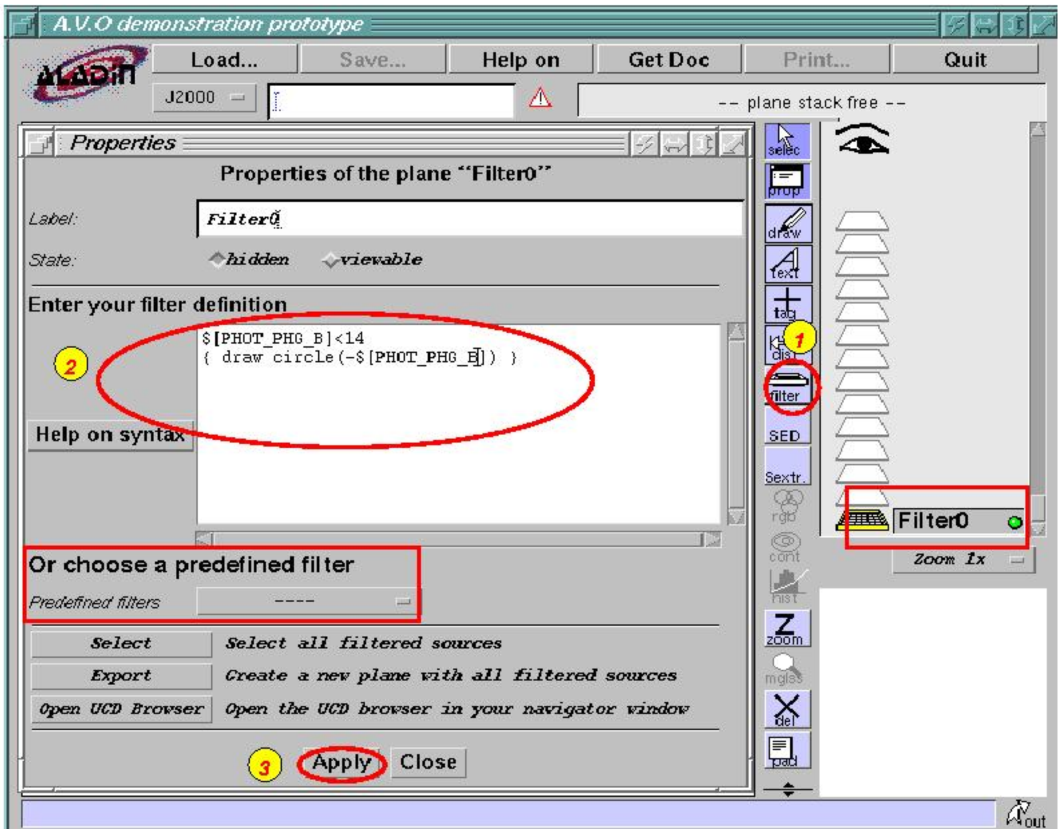
* the *line* function draws a line between 2 points on the sky. This might be quite useful when analyzing some cross-match results. Parameters are right ascension and declination of the 2 positions. All 4 parameters are assumed to be expressed in decimal degrees.
  E.g:

  ```
  {
  # Draw a line between 2 positions
  # according to given field values in the catalogue
  draw line( ${ra1}, ${dec1}, ${ra2}, ${dec1} )
  }
  ```

3 Usage in Aladin

3.1 Creating a filter

Figure 3 describes how to create a filter. First, click on te Filter button in the tool bar. This will create in the stack a new plane dedicated to a filter.



Label: Filter0

State: hidden viewable

```
$[PHOT_PHG_B]<14
{ draw circle(-$[PHOT_PHG_B]) }
```

Figure 3: Creating a filter in 3 steps

The *Properties* window will pop up, in order to let you enter your definition. Type in your filter definition, and press the *Apply* button.

In the properties window of a filter, predefined filters are intended to help you understand the syntax of a filter.
If you right-click in the definition text area, a pop-up help menu will appear and allows easy access to columns/UCDS names and list of actions.

3.2 Modify a filter

To modify a filter, open the *Properties* window of the filter. Then, modify the definition and press *Apply* to confirm the changes.
If the filter is active, the result will be updated. If it is not, the new definition will be taken into account as soon as the filter becomes activated.

3.3 Syntax errors

When creating or modifying a filter, you may enter a definition which is syntactically incorrect. If it happened, the filter is deactivated, a message pops up to let you know what the error is. At the same time, the status pastille next to the filter name in the stack becomes red.

If you try to activate a filter whose definition is incorrect, you will see a message asking you to correct the error.

3.4 Activate/Deactivate a filter

Filters are like other planes : you can easily activate or deactivate them, just by clicking on the logo.

3.5 Scope of a filter

A filter applies to all *active* catalogue planes located *below it.* If the filter is *in a folder*, it applies to all *active* catalogues located *below it* and *in the same folder*, including catalogues being in subfolders.

Figure 4 explains this principle. We have created 2 filters, *TEST* and *CIRCLE*. *TEST* just prints the string **TEST**, *CIRCLE* draws a circle according to the magnitude of the object.
As you can see, the filter *TEST* applies to planes *GSC1.2* and *USNO2* which are below it, but does not apply to plane *Simbad*. *CIRCLE* applies to *GSC1,2*, which is in the same folder, but does not apply to *USNO2* even though it is located below *CIRCLE*.

Whenever a filter or a catalogue plane has moved, whenever a new catalogue plane is created, filters results are automatically reprocessed and updated.

3.6 Applying multiple filters

You can apply several filters simultaneously. Each filter has its own scope, and each filter performs its actions apart from each other.

3.7 Creating a filter in script mode

You can create filters in script mode, via in-line commands.
The syntax is almost identical to the one you use in graphical mode. The only difference is that you have to begin with `filter filtername {` and to finish with a closing bracket `}`. The filter definition can be on multiple lines : once you entered `filter filtername {`, the prompt becomes `Aladin - Filter def.` and waits until you enter the final `}`.
E.g:

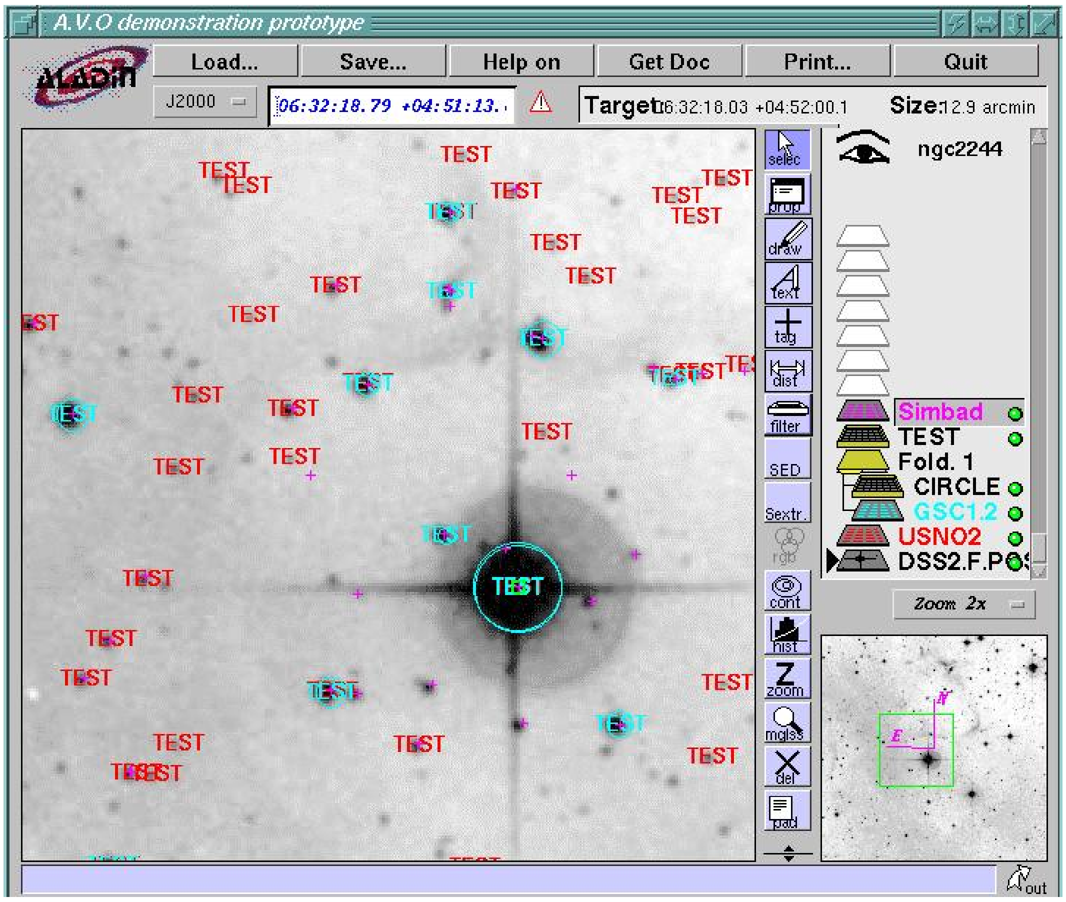


Figure 4: Scope of filters

```
Aladin> filter circle {
Aladin - Filter def.> {draw circle(-$[PHOT*])}
Aladin - Filter def.> }
```

Filters can also be activated/deactivated with the command:

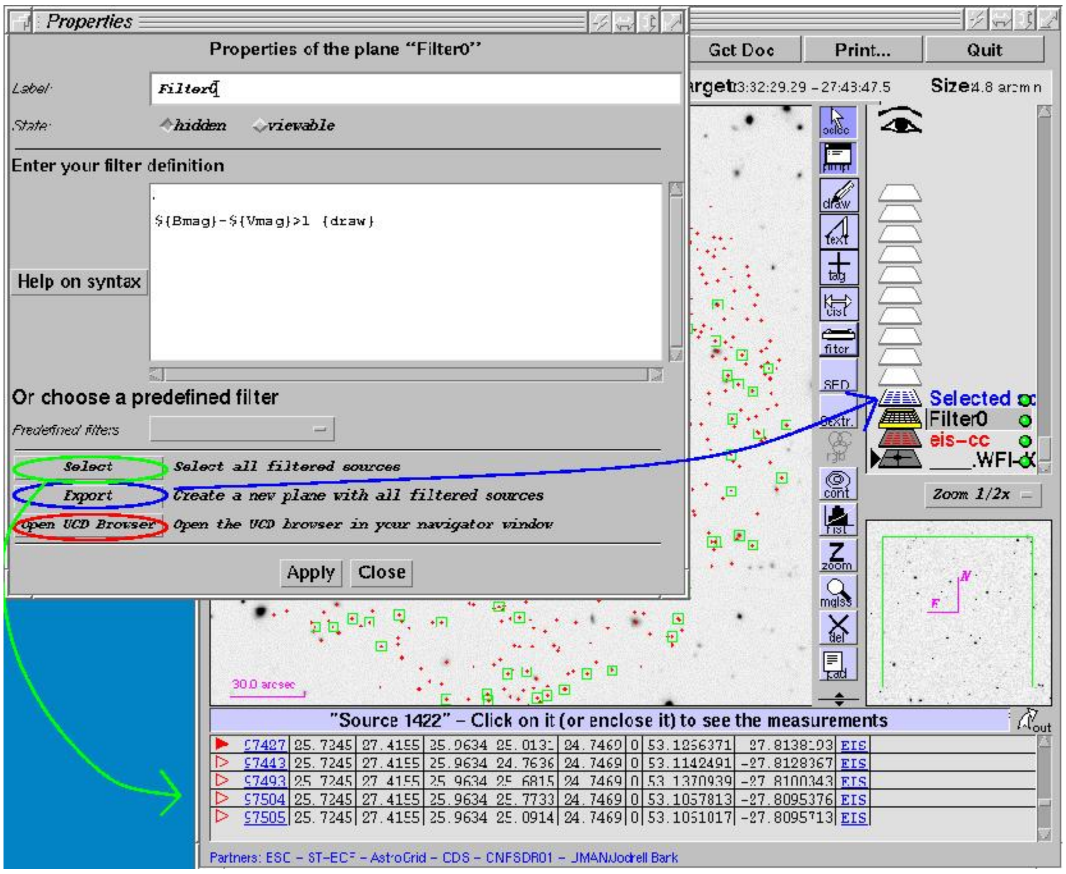
```
filter [filtername] [on|off]
```

3.8 Miscellaneous

Figure 5 shows 3 highlighted buttons contained in the *Properties* window of a filter.

- *Export* creates a new catalogue plane having all filtered sources. This can be useful to save the result of a filtering process.

This feature can be used regardless of the activated/deactivated status of the filter.



Label: Filter0

Enter your filter definition

```
${Bmag}-${Vmag}>1 {draw}
```

Select	Select all filtered sources
Export	Create a new plane with all filtered sources
Open UCD Browser	Open the UCD browser in your navigator window

Target 03:32:29.29 -27:43:47.5

"Source 1422" – Click on it (or enclose it) to see the measurements

▶	S7427	25.7245	27.4155	25.9634	25.0131	24.7469	0	53.1256371	-27.8138193	EIS
▷	S7443	25.7245	27.4155	25.9634	24.7636	24.7469	0	53.1142491	-27.8128367	EIS
▷	S7493	25.7245	27.4155	25.9634	25.6815	24.7469	0	53.1370939	-27.8100343	EIS
▷	S7504	25.7245	27.4155	25.9634	25.7733	24.7469	0	53.1057813	-27.8095376	EIS
▷	S7505	25.7245	27.4155	25.9634	25.0914	24.7469	0	53.1051017	-27.8095713	EIS

Partners: ESO – ST-ECF – AstroGrid – CDS – CNFSDR01 – JMAN/Jodrell Bank

Figure 5: Select and Export features

Appendices

A Backus Naur Form of filter syntax

This is an attempt to describe the syntax of a filter in the extended Backus-Naur-Form. It describes in a shorter way what has been explained in section 2.

```
<filter> ::= <constraints block>+

<constraints block> ::= <constraint> "{" (<action><action separator>)* "}"

<action separator> ::= Carriage Return | ";"

<constraint> ::= ( <simple condition> [ <logical operator> <constraint> ] )
               | "undefined(" (<UCD>|<column>) ")"
```

```
<condition> ::= <expression> <comparison operator> <value>

<expression> ::= ( ["+"|"-"] (<UCD>|<column>|<numeric>)
                 [ ( ("+"|"-"|"*"|"/") <expression>  | "^" <numeric> ) ] )
                 | <function>"("<expression>")"

<function> ::= abs | cos | deg2rad | exp | ln | log | rad2deg | sin | tan

<value> ::= <string> | ( <numeric> [<unit>] )

<logical operator> ::= "&&" | "||"

<comparison operator> ::=  "==" | "!=" | ">" | ">=" | "<" | "<="

<action> ::=
"draw" [ <color function> ] [ (<shape function>|<UCD>|<column>|<string>) ]
| "hide"

<UCD> ::= "$[" UCD name "]"

<column> ::= "${" column name "}"

<shape function> ::= (("circle"|"fillcircle"|"fixedcircle")"("
                       <expression> [","<numeric>","<numeric>]")")
                   | "ellipse(" <expression> "," <expression> "," <expression> ")"
                   | "pm(" <expression> "," <expression> ")"
                   | "rectangle(" <expression> "," <expression> "," <expression> ")"
                   | "line(" <expression> "," <expression> ","
                                <expression> "," <expression> ")"
                   | "square" | "rhomb" | "cross" | "plus" | "dot" | "microdot"

<color function> ::= "rgb(" <expression>, <expression>, <expression> ")"
                   | "rainbow(" <expression> ")"
                   | "black" | "blue" | "cyan" | "darkGray" | "gray" | "green"
                   | "lightGray" | "magenta" | "orange" | "pink" | "red" | "white"
                   | "yellow"
```