Filter Object API

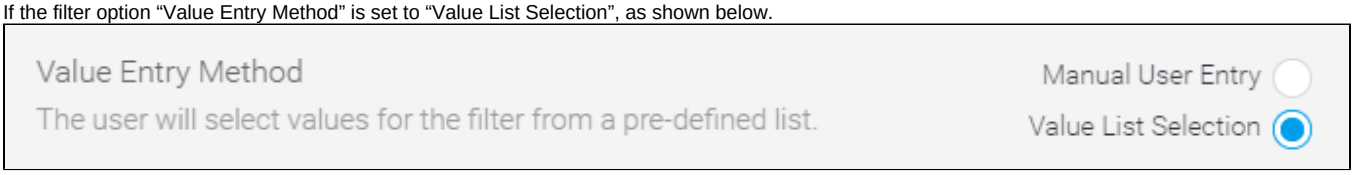
Filter Object

A filter object represents an individual filter. It can be fetched using the *getFilter* function in the Filters API.

It has a number of properties, listeners and functions that allow developers to create code that manipulates the filter and reacts to events that happen on the filter itself.

Possible Filter Values

If the filter option “Value Entry Method” is set to “Value List Selection”, as shown below.



In this case, Yellowfin will attempt to generate possible values for the filter. These values can be retrieved from Reference Code Values, Cached Values or a Custom Query.

The front-end filter options will be passed an Array of JavaScript objects that looks like this:

```
[{
    value: 'AU', //The value that is actually applied to any queries
    description: 'Australia', //The human readable version of the value
},{
    value: 'NZ',
    description: 'New Zealand'
}]
```

The “value” property is the value that will be applied to any queries that are run using this filter. Using SQL as an example, if I was to select the first value in the above list, an SQL query like this could be generated:

```
SELECT Country FROM CountryTable where Country = 'AU'
```

The “description” property is the value that will be displayed to a user. It will not be used as part of any filter queries. It is possible for the value and description to be the same.

Property Reference

name

Returns

String - The name of the filter.

Description

Returns the display name of the filter, this is the name that will appear when the filter is rendered in the Yellowfin UI.

Read Only

Yes.

operator

Returns

String - Uppercase operator of the filter. This is the operator used by Yellowfin to determine what SQL operator to use when building a query.

Read Only

Yes.

Possible Values

- EQUAL - Equal to
- NOTEQUAL - Different from
- GREATER - Greater than
- GREATEREQUAL - Greater than or equal to
- LESS - Less than
- LESSEQUAL - Less than or equal to
- BETWEEN - Between
- NOTBETWEEN - Not Between
- INLIST - In List
- NOTINLIST - Not In List

Example

Shows the operator of the filter Demographic, in this case INLIST :

```
let filter = filter.get('Demographic');
console.log(filter.operator); //outputs 'INLIST'
```

displayOperator

Returns

String - Translated and human readable version of the filters operator.

Read Only

Yes.

Possible Values

- EQUAL - Equal to
- NOTEQUAL - Different from
- GREATER - Greater than
- GREATEREQUAL - Greater than or equal to
- LESS - Less than
- LESSEQUAL - Less than or equal to
- BETWEEN - Between
- NOTBETWEEN - Not Between
- INLIST - In List
- NOTINLIST - Not In List

Example

```
let filter = filter.get('Demographic');
console.log(filter.displayOperator); //outputs 'In List'
```

Notes/Limitations

This is intended as a helper property to get a human readable version of the operator, if you are writing any logic that depends on the operator, use *filter.operator* instead.

uuid

Returns

String - UUID of the filter.

Read Only

Yes .

possibleValues

Returns

Array [Object]

Description

An array of the possible value objects that the filter renderer uses to show values for a user to select.

```
[{
    value: 'AU', //The value that is actually applied to any queries
    description: 'Australia', //The human readable version of the value
},{
    value: 'NZ',
    description: 'New Zealand'
}]
```

This property will only return values in cases where the filter property "Value Entry Method" is set to "Value List Selection" when creating the filter. If this is set to anything other than "Value List Selection" this property will return null.

Example

Use the possibleValues parameter to select all the values for a filter.

```
let possibleValues = filter.possibleValues; //Get the possible values out of the filter
let newFilterValues = []; //Create a new array that will be applied to the valueList function
possibleValues.forEach(valueObject => { //Iterate over the possibleValues and then push the value property to the newFilterValues array
    newFilterValues.push(valueObject.value);
});

filter.setValueList(newFilterValues, true);
```

between

Returns

Boolean.

Description

Returns true if the filter has a “Between style” operator, this can be “Between” or “Not Between”. This can be used if you want to determine whether there should be multiple input elements if you are writing a custom input element. This should not be used as a replacement for checking if the operator of the filter is BETWEEN.

Read Only

Yes.

list

Returns

Boolean.

Description

Returns true if the filter has a list operator. The list operators are “In List” and “Not In List”. In List and Not In List share a lot of functionality so this can be useful if you need to determine that functionality should be enabled.

Read Only

Yes.

values

Returns

Object.

Description

Returns an object with the currently staged values in an Object which can contain the following properties:

- valueOne
- valueTwo
- valueList

What is contained in this object will depend on the operator of the filter. It will only return values that are relevant to that type of filter.

A list filter should only contain “valueList”.

A single entry filter should only contain “valueOne”.

A between filter should contain “valueOne” and “valueTwo”.

Read Only

Yes.

Example

'In List' example:

```
let filter = filters.getFilter('Demographic');
filter.setValues(['Adventure']);

console.log(filter.values); //Outputs { valueList: ['Adventure'] }
```

'Between' example:

```
let filter = filters.getFilter('Age at Camp');
filter.setValueOne(15);
filter.setValueTwo(35);
console.log(filter.values); //Outputs { valueOne: 15, valueTwo: 35 };
```

'Single Entry' example:

```
let filter = filters.getFilter('Age at Camp');
filter.setValueOne(15);
console.log(filter.values); //Outputs { valueOne: 15 };
```

valueOne

Returns

String or Number.

Description

Returns the currently staged valueOne on the filter. If the filter is a list filter this will return null.

Read Only

True.

Example

```
let filter = filters.getFilter('Average Age at Camp');
filter.setValueOne(35);
console.log(filter.valueOne); //Outputs '35'
```

valueTwo

Returns

String or Number.

Description

Returns the currently staged valueTwo on the filter. If the filter is a list filter or a single entry filter this will return null.

Read Only

True.

Example

```
let filter = filters.getFilter('Average Age at Camp');
filter.setValueTwo(65);
console.log(filter.valueTwo); //Outputs '65'
```

valueList

Returns

Array{String/Number}

Description

Returns the currently staged valueList values on the filter. For 'Single Entry' and 'Between' filters this will return null.

Read Only

True.

Example

```
let filter = filters.getFilter('Demographic');
filter.setValueList['Adventure', 'Luxury']);
console.log(filter.valueList); //Outputs ['Adventure', Luxury']
```

appliedValues

Returns

Object.

Description

Returns the currently applied filter values in an Object which can contain the following properties:

- valueOne
- valueTwo
- valueList

What is contained in this object will depend on the operator of the filter. It will only return values that are relevant to that type of filter.

A list filter should only contain “valueList”.

A single entry filter should only contain “valueOne”.

A between filter should contain “valueOne” and “valueTwo”.

Read Only

Yes.

Example

In List example:

```
let filter = filters.getFilter('Demographic');
filter.setValues(['Adventure'], true); //Set the value and apply

console.log(filter.appliedValues); //Outputs { valueList: ['Adventure'] }
```

Between example:

```
let filter = filters.getFilter('Age at Camp');
filter.setValueOne(15);
filter.setValueTwo(35);
filter.applyFilter(); //Apply the values
console.log(filter.appliedValues); //Outputs { valueOne: 15, valueTwo: 35 };
```

Single Entry example:

```
let filter = filters.getFilter('Age at Camp');
filter.setValueOne(15, true);
console.log(filter.appliedValues); //Outputs { valueOne: 15 };
```

appliedValueOne

Returns

String or Number.

Description

Returns the filters currently applied valueOne. If the filter is a list filter, this will return null.

Read Only

True.

Example

```
let filter = filters.getFilter('Average Age at Camp');
filter.setValueTwo(35, true); //Set valueTwo and apply it
console.log(filter.appliedValueOne); //Outputs '35'
```

appliedValueTwo

Returns

String or Number.

Description

Returns the filters currently applied valueTwo. If the filter is not a between filter, this will return null.

Read Only

True.

Example

```
let filter = filters.getFilter('Average Age at Camp');
filter.setValueTwo(65, true); //Set valueTwo and apply it
console.log(filter.appliedValueTwo); //Outputs '65'
```

appliedValueList

Returns

Array{String/Number}

Description

Returns the filters currently applied *valueList*. For Single Entry and Between filters this will return null.

Read Only

True.

Example

```
let filter = filters.getFilter('Demographic');
filter.setValueList['Adventure', 'Luxury']);

filter.applyFilter();
console.log(filter.appliedValueList); //Outputs ['Adventure', Luxury']
```

defaultValues

Returns

Object.

Description

Returns the filters default values in an Object which can contain the following properties:

- valueOne
- valueTwo
- valueList

What is contained in this object will depend on the operator of the filter. It will only return values that are relevant to that type of filter. If a between filter has a lower value set up as a default, but not an upper value, *valueOne* will be set and *valueTwo* will be null.

A list filter should only contain “valueList”.

A single entry filter should only contain “valueOne”.

A between filter should contain “valueOne” and “valueTwo”.

Read Only

Yes.

Example

In List example:

```
let filter = filters.getFilter('Demographic');
console.log(filter.defaultValues); //Outputs { valueList: ['Adventure'] }
```

Between example:

```
let filter = filters.getFilter('Age at Camp');
console.log(filter.defaultValues); //Outputs { valueOne: 15, valueTwo: 35 };
```

Single Entry example:

```
let filter = filters.getFilter('Age at Camp');
console.log(filter.defaultValues); //Outputs { valueOne: 15 };
```

defaultValueOne

Returns

String or Number.

Description

Returns the filters default valueOne. This will only return anything if the filter has had a default value set while the content was being created. Will return null if the filter is a list type.

Read Only

True.

Example

```
let filter = filters.getFilter('Average Age at Camp');
console.log(filter.defaultValueOne);
```

defaultValueTwo

Returns

String or Number.

Description

Returns the filters default valueTwo. This will only return anything if the filter has had a default value set while the content was being created. Will always return null if the filter is not a between type.

Read Only

True.

Example

```
let filter = filters.getFilter('Average Age at Camp');
console.log(filter.defaultValueTwo);
```

defaultValueList

Returns

Array{String/Number}

Description

Returns the filters default valueList. This will only return anything if the filter has had a default value set while the content was being created. Will always return null if the filter is not a list type.

Read Only

True.

Example

```
let filter = filters.getFilter('Demographic');
console.log(filter.defaultValueList);
```

Function Reference

reset(apply)

Returns

Nothing.

Description

Resets the current filter to its default values. If the apply parameter is true, this will also immediately apply the filter to the report or dashboard that it is present on.

Parameter

Apply - Boolean - Default: True

Example

The filter Athlete Country has the default values of ['AU', 'NZ']

```
filter.setValue(['UK', 'US']);

console.log(filter.values); //Outputs {valueList: ['UK', 'US'] }

filter.reset();

console.log(filter.values); //Outputs {valueList: ['AU', 'NZ']
```

clear(apply)

Returns

Nothing.

Description

Clears the values from the filter and runs the report/dashboard if apply is true. All values will be set to undefined.

Parameters

Apply - Boolean - Default: True

Example

The filter Athlete Country has the default values of ['AU', 'NZ']

```
filter.setValue(['UK', 'US']);

console.log(filter.values); //Outputs {valueList: ['UK', 'US'] }

filter.clear();

console.log(filter.values); //Outputs {valueList: null]}
```

resetToLastAppliedState()

Returns

Nothing.

Description

Resets the filter values from their current values to whatever is in the applied values object. This can serve as an "undo" function as it allows users to return their filters to the last state that the content was run in.

Example

```
filter.setValue(['UK', 'US']);
filter.apply(); //Set the value on the report

filter.setValue(['DE']);
console.log(filter.values); //Outputs {valueList: ['DE'] }

filter.resetToLastAppliedState();

console.log(filter.values); //Outputs {valueList: ['UK', 'US']}
```

setValueOne(value, apply)

Returns

Nothing.

Description

Sets the value of valueOne; will also apply and run the report/dashboard if apply is true.

Null is allowed as a value, from a user perspective this would just be clearing valueOne.

If setValueOne is called when using a list operator (In List or Not In List) this entire call will do nothing.

Parameters

value - String, Number

apply - Boolean - Default: false

Example

```
filter.setValueOne('Relaxation'); //Change the value to Relaxtion but don't immediately run the report

console.log(filter.valueOne); //Should return 'Relaxation'
```

setValueTwo(value, apply)

Returns

Nothing.

Description

Sets the value of valueTwo; will also apply and run the report/dashboard if apply is true.

Null is allowed as a value, from a user perspective this would just be clearing valueTwo.

If setValueTwo is called when using an operator that isn't Between or Not Between this function call will do nothing.

Parameters

value - String, Number

apply - Boolean - Default: false

Example

```
//Filter is a between filter (Average Age at Camp)
filter.setValueOne(15);
filter.setValueTwo(35);

console.log(filter.valueOne + " to " + filter.valueTwo); //Should output "15 to 35"
```

setValueList(valueList, apply)

Returns

Nothing.

Description

Sets the valueList property of the values objects. Can also immediately apply the filter to the report/dashboard if the *apply* parameter is true.

When used on a list operator, this will apply the array as it is to the filter.

If this is used on a single value operator, valueOne will be set to the first value in the array.

If this is called on a filter with a between operator, valueOne will be set to the first value in the array and valueTwo will be set to the second value in the array.

Parameters

valueList - Array

apply - Boolean - Default: false

Example

```
//Set the value of the filter "Demographic" to be Adventure, Family, Sport
filter.setValueList(['Adventure', 'Family', 'Sport']);

//Set the values and immediately apply the filter
filter.setValueList(['Adventure', 'Relaxation'], true);

//Can also be used to set the between values in Average Age at Camp
filter.setValueList([15, 35]);

filter.setValueList([15]);
```

setValue(value, apply)

Returns

Nothing.

Description

Catch all functions that attempt to apply the passed value to the correct location. If *apply* is true then it will also immediately apply to the report.

There are a number of different things that can occur when passing in different types of “value”.

If *value* is a Number or a String.

A *list* filter will treat this as you selecting a single value. It is equivalent to calling:

```
filter.setValueList(['single value']);
```

A between or single entry filter will set valueOne to the passed value. Effectively this:

```
filter.setValueOne('first value');
```

If *value* is an Array,

A list filter will apply the array directly to its valueList attribute.

A between or single entry filter will take the first value from the Array and apply it to the filters valueOne attribute.

A between filter will take the second value from the Array and apply it to the filters valueTwo attribute.

```
filter.setValue([15, 35]);
```

On a between filter will yield the same result as:

```
filter.setValueOne(15);
filter.setValueTwo(35);
```

If *value* is an Object,

Between and single entry filter will look for a “valueOne” attribute on that object and use that to populate the filters valueOne attribute.

Between filters will also look for a “valueTwo” attribute on the object and use that value to populate the filters valueTwo attribute.

List filters will look for a valueList attribute and use that to populate the filters valueList attribute.

Parameters

value- String, Number, Array, Object

apply - Boolean - Default: false

Example

Using on a between filter with an object:

```
let filter = filters.getFilter('Age at Camp');
filter.setValue({
valueOne: 15, //Sets valueOne to 15
valueTwo: 35 //Sets valueTwo to 35
valueList: [1,2,3,4,5]  //Ignored
});

filter.setValue({
valueOne: 15, //Sets valueOne to 15
valueTwo: 35 //Sets valueTwo to 35
});
```

Using on a list filter with an object and an array

```
let filter = filters.getFilter('Demographic');
filter.setValue(['Adventure']); //Using an array

filter.setValue({ valueList: ['Adventure'] }); //Using an object
```

applyFilter()

Returns

Nothing.

Description

Copies the filter's currently staged values, *valueOne*, *valueTwo* and *valueList to* the applied equivalents of those objects. If there are any changes between *valueOne* and *appliedValueOne*, *valueTwo* and *appliedValueTwo* and *valueList* and *appliedValueList,* an 'applied' event will be triggered with the values that have changed. See applied event for details.

Example

Apply the value Adventure to the Demographic filter.

```
let filter = filters.getFilter('Demographic');
filter.setValue(['Adventure']);
filter.applyFilter();
```

setPossibleValues(values)

Returns

Nothing.

Description

Updates the "possibleValues" of the filter to the passed Array. If an Array is passed then the filter will update its values and the Yellowfin UI widget will be updated accordingly. If *null* is passed then the possible values will be cleared. If any other type of object is passed it will be ignored.

The passed Array must be an Array of Objects with "value" and "description" properties.

The "value" property is what will be applied as part of the SQL query when the filter value is selected. The "description" property is what will be displayed to the user. In a lot of cases these might be the same. However you can use the "description" property to account for translations, or make a human readable version of the value.

```
[{
    value: {Number, String},
    description: {String}
}]
```

The values will be displayed in the order that they are added to the list.

Example

Add an extra Demographic value to the current filter options for the filter Demographic:

```
let possibleValues = filter.possibleValues;
possibleValues.push({
    value: 'Relaxation',
    description: 'Relaxation'
});
filter.setPossibleValues(possibleValues);
```

Override the entire possible values array with values of your own:

```
let possibleValues = [];
possibleValues.push({
    value: 'FIRST', //The data that is stored in the table for this filter is upper case.
    description: 'First' //Upper case can be painful to read, so put a more readable version to be displayed in the description
});

possibleValues.push({
  value: 'SECOND',
  description: 'Second'
});

filter.setPossibleValues(possibleValues);
```

Notes/Limitations

If the filter is part of a filter hierarchy any values you set through this property will be overwritten when the parent filter changes and pushes new values into this filter.

addEventListener(eventName, callbackFunction)

Returns

Number.

Description

Creates a listener on an event which will call the *callbackFunction* whenever that particular event occurs.

When an event is set up a unique ID is assigned to it which is returned as the result of this function. This ID can be used by the *removeEventListener* functi on to remove the callback when you are done with it. If you are writing an application that requires loading and unloading reports it is recommended that you keep track of these listenerIds so that you can remove them when no longer needed.

See the event reference section for details about the events that this API will trigger itself.

It is also possible for developers to trigger their own events (see .trigger()) which can also be listened to using this function.

Example

Create a listener on the changed object and remove it once the event occurs once:

```
let eventListenerId = filter.addEventListener('changed', function(event) {
    console.log(event.filter.name +  ' changed value');
    filter.removeEventListener(eventListenerId);
});
```

removeEventListener(listenerId)

Returns

Nothing.

Description

Removes the callback function associated with the passed listenerId. This will mean that when the event associated with that callback function occurs that callback will not be fired anymore.

Example

```
let eventListenerId = filters.addEventListener('changed', function(event) {
    console.log('One of my filters changed');
    filters.removeEventListener(eventListenerId);
});
```

trigger(eventName, eventData)

Returns

Nothing.

Description

Triggers an event on the FilterObject and calls any listener functions that have been created for that event. This can be used to trigger custom events that you may have set up for the application.

Example

When using a custom filter input we could trigger a filter click event, so that another part of the application could react to that.

```
//Add a 'userClick' listener to the filter object, which we will set up a trigger for later on.
filters.addEventListener('userClick', function(event) {
    console.log('A user clicked on the element ' + event.element + ' which is tied to this filter');
});
//Get the custom filter list from the DOM and create a click listener on that which will trigger userClicked events on the filter
let myCustomFilterList = document.querySelector('div#customFilterList')
myCustomFilterList .addEventListener('click', function(e) {
    filters.trigger('userClicked', { element: e.currentTarget } );
});
```

Event Reference

There are a number of events that will trigger on a filter object when a user or piece of code takes a certain action.

Any event that is triggered through the *FilterObject* will have an object that contains the filterUUID and the filter object itself.

```
filter.addEventListener('changed', function(event) {
      console.log(event.uuid); //The filters UUID that the event was triggered from
      console.log(event.filter);//The FilterObject that the event was triggered from
});
```

changed

Description

Occurs when any of the filter values changes. Triggers with an object that contains the values that changed as well as the previous values of all of those values.

Parameters

Event - Object

Contains:

- uuid - String - UUID of the filter that triggered the event.
- filter - FilterObject - FilterObject of the filter that triggered the event.
- changed - Object - Object containing any values that have changed. Could possibly contain valueOne, valueTwo or valueList.
- Previous - Object - Object containing the old values of all of the changed values. Could possibly contain valueOne, valueTwo or valueList.

Example

```
filter.addEventListener('applied', function(event) {
      console.log(event.filter.name + " has just been applied with the following changed values " + JSON.stringify(event.changed));
});
```

applied

Description

Occurs when the filter is applied to the linked piece of content (report/dashboard). And the applied values change.

Parameters

Event - Object

Contains:

- uuid - String - UUID of the filter that triggered the event.
- filter - FilterObject - FilterObject of the filter that triggered the event.
- changed - Object - Object containing any values that have changed. Could possibly contain valueOne, valueTwo or valueList.
- Previous - Object - Object containing the old values of all of the changed values. Could possibly contain valueOne, valueTwo or valueList.

Example

```
filter.addEventListener('applied', function(event) {
    console.log(event.filter.name + " has just been applied with the following changed values " + JSON.stringify(event.changed));
});
```

Notes/Limitations

The applied event may be triggered in a case like this:

```
let filter = filters.getFilter('Demographic');
filter.setValue(['Adventure']);
filter.apply();

filter.setValue(['Adventure']);
filter.apply();
```

Due to the code creating a new Array when calling setValue. So any comparison that occurs will be of Array to Array.

The correct values will still be applied to the report.

reset

Description

Occurs when the reset function is called on the filter. This can occur from user interaction, clicking the reset button on a filter menu, or by a developer explicitly calling it.

Example

```
filter.addEventListener('reset', function(event) {
    console.log(event.filter.name + " has just been reset");
});
```

Parameters

Event - Object

Contains:

- uuid - String - UUID of the filter that triggered the event.
- filter - FilterObject - FilterObject of the filter that triggered the event.

cleared

Description

Occurs when the clear function is called on the filter. This can occur from user interaction, or by a developer explicitly calling it.

Difference between cleared and reset

Resetting a filter resets it to its default values. Where as clearing a filter will set it to have no values.

Parameters

Event - Object

Contains:

- uuid - String - UUID of the filter that triggered the event.
- filter - FilterObject - FilterObject of the filter that triggered the event.

Example

```
filter.addEventListener(cleared, function(event) {
    console.log(event.filter.name + " has just had its values cleared");
});
```