Report API

- Overview of Report API
  - Default Chart ID & Chart IDs
    - Field IDs
    - Drill
  - Property Reference
    - name
    - reportUUID
    - entityUUID
    - filters
    - fields
    - charts
    - reportKey
  - ReportAPI Parameters
    - useDefaultAssistedInsightsPanel
    - preventDefaultAssistedInsights
- Function Reference
  - getFieldId(fieldName)
  - getFieldName(fieldId)
  - getChartId(chartId)
  - applySeriesSelection(series, chartId)
  - createReportElement(elementOptions)
    - Interactions
    - Example
  - sortAscending(fieldId)
  - sortDescending(fieldId)
  - runReport()
  - runAssistedInsights
    - Example
  - openComparePanel(value, comparableValues, metricDescription)
  - displayAssistedInsightsData(assistedInsightsData)
  - cancelAssistedInsights(assistedInsightsUUID)
  - deleteTemporaryAssistedInsightsReport(assistedInsightReportId)
  - registerOutputType(outputOptions, callback)
  - removeOutputType(outputKey)
  - reset()
  - isDrillAnywhere()
  - isDrillDown()
  - isDrillDownApplied()
  - isDrillAnywhereApplied()
  - drill(fieldId, value, toField)
  - drillUpLevels(fieldId, levels)
  - drillUpOneLevel()
  - drillReset(fieldId)
  - timeslider(from, to, chartId)
  - unitSelection(unit, chartId)
  - convertDrillDownToArray() & converDrillAnywhereToArray()
- Event Reference
  - seriesSelection
  - sort
  - reportStart
  - reportComplete
  - assistedInsightsRequestedByChart
  - assistedInsightsStarted
  - compareValuesSelected
  - assistedInsightsInProgress
  - assistedInsightUpdate
  - assistedInsightsCompleted
  - assistedInsightsCancelled
  - assistedInsightsError
  - assistedInsightsExceptionError
  - drilldown
  - drillanywhere
  - drillUp
  - reportRunAlreadyInProgress
  - resetDrill
  - reset
  - slider
  - unitSelection

Overview of Report API

The Report API controls the state of the report. It contains information about the current state of the report, including the current drill state, sort criteria, filter values and other report interactions that may have been applied.

The Report API also controls the actual running of the report and will submit any of the interactions to the Yellowfin server when the report is run. The API also triggers a number of events when actions on the report are taken, either by a user action or an automated developer action.

Default Chart ID & Chart IDs

A report can have a number of charts; each of those charts will have a *chartId* and *chartUUID*. The *chartId* will change when a report is edited and published again, whereas the *chartUUID* will always remain the same. All functions that require a *chartId* can accept a *chartUUID* as well, however it is preferred to use *chartUUID* on any function that requires a *chartId* to be passed as it will never change.

There is also a “Default Chart” which will be the first usable chart on the report. A usable chart is one that has fields assigned to it. When a function requiring a *chartId* has no *chartId* passed to it, the default chart would be used.

You can access all the charts on a report using the *charts* property.

Field IDs

A report will have a number of fields; each of these fields will have a unique ID within the context of that report. This ID will never change, even when the report is edited. There is a function that allows you to retrieve the fieldId based on the field’s name, *getFieldId(fieldName)*. Most functions that accept a *field Id* will also be able to accept a field name and will attempt to find the *fieldId* based on the passed name.

You can access all of the fields on a report using the *fields* property.

Drill

When Drill Anywhere or Drill Down is applied to a field the process is as follows:

1. Apply the drill value as a hidden filter.
2. Replace the field we were drilling down on with the field we are drilling to.
3. Run the query with the new field and filter.

Using a very simple drill example, consider a report that just has one column: “Agency Type”. In the Ski Team view (bundled with the Tutorial content in Yellowfin); “Agency Type” has a Drill Hierarchy setup with “Agency Name” as its child. When drill down is applied to the “Agency Type” field the report logic knows to drill down to the ‘Agency Name” field.

When the report is first run the SQL will look like this:

```
SELECT DISTINCT
      "TRAVELAGENCY"."AGENCYTYPE" AS C1
FROM "TRAVELAGENCY"
```

After drill down is applied on ‘Agency’ value you can see that Agency Type has been replaced in the SQL and the Agency Type filter has also been added.

```
SELECT DISTINCT
      "TRAVELAGENCY"."AGENCYNAME" AS C1
FROM "TRAVELAGENCY"
WHERE (
     "TRAVELAGENCY"."AGENCYTYPE" = 'Agency'
)
```

Drill Anywhere works the same way, except it doesn’t have a predefined “drill to” field and the user (or developer if using an API call) defines which field they need to drill to.

top

Property Reference

name

Returns

String

Description

Returns the report’s name. This will be translated into the user's language if translations are enabled and a translation has been provided for that language.

reportUUID

Returns

String

Description

Returns the UUID of the report.

entityUUID

Returns

String

Description

The Unique Identifier for this instance of the report. This will be defined when the report is included on a Dashboard. Otherwise this will be null.

filters

Returns

FiltersAPI

Description

Returns the FiltersAPI for the Report. If this report is included on a dashboard, the FiltersAPI will be the same as the Dashboard’s FiltersAPI.

fields

Returns

Array[ReportField]

Description

Returns an ordered Array of the report’s fields.

charts

Returns

Object{String, ReportChart}

Description

Returns an Object containing all of the ReportCharts on the Report, keyed by chart UUID.

reportKey

Returns

String

Description

A unique identifier for the report that is used to identify exactly which instance of a report object should be used. A number of events will contain this as part of the information.

top

ReportAPI Parameters

These parameters should be set individually for every report that requires them. For example, Assisted Insights is usually triggered from a chart, so these parameters would need to be set on the parent report's API.

useDefaultAssistedInsightsPanel

Default is true.

This indicates to the system that the default Assisted Insights display panel should be rendered. Must be false if *displayAssistedInsightsData* is being overwritten with a custom panel.

Example

In Code Mode:

```
let reportElement = this.apis.canvas.select('Performance by Region');

reportElement.onReportLoad.then(() => {

let reportAPI = reportElement.reportAPI;

reportAPI.useDefaultAssistedInsightsPanel = true;

});
```

preventDefaultAssistedInsights

Default is false.

Stops the usual default Assisted Insights flow so you can call *runAssistedInsights* when required.

This is particularly useful if you wish to customise the UI elements of the Assisted Insights process (loader, etc.) but still require the process to be triggered from a chart tooltip.

Example

```
let reportElement = this.apis.canvas.select('Performance by Region');

reportElement.onReportLoad.then(() => {

let reportAPI = reportElement.reportAPI;

reportAPI.preventDefaultAssistedInsights = true;

});
```

top

Function Reference

getFieldId(fieldName)

Returns

Number

Description

Returns the field ID for the passed *fieldName*. Returns null if no matching field is found.

Example

Get the fieldId of the field “Invoiced”:

```
let fieldId = report.getFieldId('Invoiced');
```

getFieldName(fieldId)

Returns

String

Description

Returns the field name for the passed *fieldId*. If no matching field is found null will be returned.

Example

Log that a particular field was drill down on:

```
report.addListener('drilldown', function(event) {
    console.log(report.getFieldName(event.eventData.fieldId) + " just had a drill value applied to it");
})
```

getChartId(chartId)

Returns

Number

Description

Returns the internal chartId for the passed chart name or chart UUID. If chartId is passed, the report’s default chart ID will be returned. If no matching chart is found, *null* will be returned.

applySeriesSelection(series, chartId)

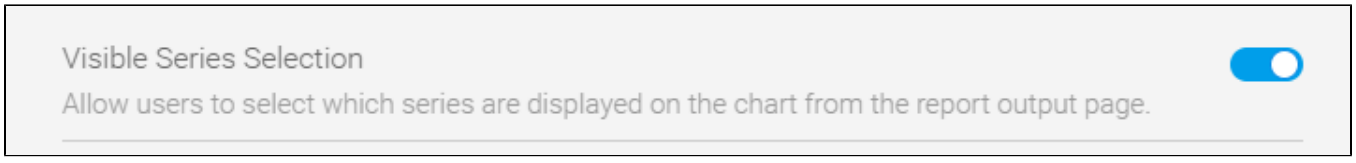
Returns

Nothing

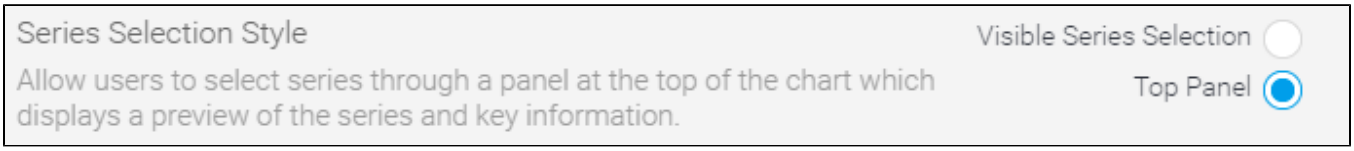
Description

Applies series selection to the passed chartId. Series Selection works by either replacing the series field on a chart with another field or adding multiple series fields to the chart to be rendered together.

The option *Visible Series Selection* is required to be turned on at the chart level for this function to have any effect on the report. See Image below:



If the option *Series Selection Style* is set to “Top Panel” or “Left Panel” (as shown below), the chart’s series selection becomes a single selection type. Meaning only a single series can be selected at a time. When this option is enabled, the chart will only take the first value out of the passed array.



If the property is set to *“Visible Series Selection”,* the type of chart will determine whether all of the values are used or not.

If a *fieldId* that is not present on the chart or report is passed to the **applySeriesSelection** function, that *fieldId* will be ignored and the chart will be set back its default series value.

Parameters

Series - Array[Number]

Array of FieldId’s that you wish to apply to the report.

chartId - String, Number

The chartId that you want to apply the series selection to. If no chartId is passed the default chart ID will be used.

Example

Add a number of fields to the default charts series selection:

```
let fieldsToAdd = ['Invoiced', 'Cost', 'Profit'];
report.applySeriesSelection(fieldsToAdd );
```

Apply using a field ID:

```
report.applySeriesSelection(5);
```

Apply using a field name:

```
report.applySeriesSelection('Invoiced');
```

createReportElement(elementOptions)

Returns

HTML Element

This method only exists within the Embedded API. It will not be available for use within Dashboard Code Mode.

It creates and returns an element which will render the visualization of a report. If an *element* is passed as part of *elementOptions*, the new report element will be appended to that as soon as it is created.

There are a number of options that can be passed along with the request to create the exact experience you want.

> **Note**
>
> You can use all the functions that BaseAPI.loadreport takes (apart from reportId, instanceName and filterValues) with createReportElement too.

Examples

Example 1: Load a report with multiple charts and then render them all into separate elements rather than having a canvas display.

HTML:

```
<div id=”reportContainer”></div>
```

CSS:

```
div.reportContainer app-report {
    width:500px;
    height:500px;
}
```

JS:

```
let reportUUID = 'a-report-uuid';
yellowfin.loadReport({
   reportId: reportUUID
}).then(report => {
    let charts = report.charts;
    Object.values(charts).forEach(chart => {
        report.createReportElement({
            element: document.querySelector('#reportContainer'),
            displayType: 'CHART', //Tell the report to render as a chart
            chartUUID: chart.uuid, //Tell the report which chart you wish to render
            displayToolbar: false //Tell the report not to render any toolbar
        });
    });
});
```

This will then create a 500x500 render for each chart within the container.

Example 2: Load a report and then render its canvas and table

HTML:

```
<div id="reportContainer"></div>
```

CSS:

```
div.reportContainer app-report {
    width:500px;
    height:500px;
}
```

JS:

```
let reportUUID = 'a-report-uuid';
yellowfin.loadReport({
    reportId: reportUUID
}).then(report => {
    report.createReportElement({
        element: document.querySelector('#reportContainer'),
        displayType: 'CANVAS', //Tell the report to render as a chart
        displayToolbar: false //Tell the report not to render any toolbar
    });
    report.createReportElement({
        element: document.querySelector('#reportContainer'),
        displayType: 'REPORT', //Tell the report to render as a chart
        displayToolbar: false //Tell the report not to render any toolbar
    });
});
```

Interactions

The interactions object enables you to define which interactions should be available on a particular visualization of a report. This lets you tell the Yellowfin renderer which functionality you want the user to be able to use within that visualization.

If there are multiple elements representing the same report and you wish to disable an interaction, you must disable it on all the elements you are displaying. For example, if you disabled drill down on the first element and then did not on the second, the user would still be able to drill down on the second element.

By default, any interaction that is not explicitly set to false is treated as enabled and will display if the report allows it.

animation

Allows chart animations to be disabled.

annotations

Allows annotations to be disabled.

brushing

Allows brushing functionality to be disabled.

drillAnywhere

Allows drill anywhere functionality to be disabled.

drillBreadcrumbs

Allows drill breadcrumb functionality to be disabled

drillDown

Allows drill down functionality to be disabled.

drillThrough

Allows drill through functionality to be disabled.

seriesSelection

Allows seriesSelection functionality to be disabled

timeSlider

Allows the timeSlider functionality to be disabled.

unitSelection

Allows unit selection functionality to be disabled.

Example

To disable all user interactivity on the report using the *createReportElement* function:

```
report.createReportElement({
    interactions: {
        animation: false,
        annotations: false,
        brushing: false,
        drillBreadcrumbs: false,
        drillDown: false,
        drillAnywhere: false,
        drillThrough: false,
        seriesSelection: false,
        timeSlider: false,
        unitSelection: false,
    }
});
```

sortAscending(fieldId)

Returns

Nothing

Description

Applies ascending sorting to the passed field. If the passed fieldId doesn't match a field on the report, no sorting will be applied.

Parameters

fieldId: Number, String

FieldId or Name of the field that we wish to sort.

sortDescending(fieldId)

Returns

Nothing

Description

Applies a descending sort to the passed field. If the passed fieldId doesn't match a field on the report, no sorting will be applied.

Parameters

fieldId: Number, String

FieldId or Name of the field that we wish to sort.

runReport()

Returns

Nothing

Description

Runs the report with the current state of the report applied. If the *runReport* function is triggered multiple times in a short period of time, the run report request will only be sent to the server once.

Example

Re-run the report every 5 seconds after the report finishes running.

```
//Add a listener for the reportComplete event
report.addListener('reportComplete', () => {
        //Add a 5000ms delay before triggering runReport again
       setTimeout(() => {
              report.runReport();
       }, 5000)
});

report.runReport();
```

runAssistedInsights

Returns

Promise

Description

Begins the Assisted Insights process. This will generate a temporary Assisted Insights report and resolve the promise with the report data once it has completed. The assistedInsightsCompleted event will also be triggered with the same data.

> **Note**
>
> A new Assisted Insights report is generated each time this is called, even if the options are the same. If you are calling this manually, we recommend safeguarding against duplicate requests or multiple requests being made until the previous ones have returned.

The easiest way to generate data in the required format for this function is by triggering the Assisted Insights process on an existing chart and using the eventData from the `assistedInsightsRequestedByChart` event.

Parameters

Options {object}

- loader {DOM element} (Optional). A custom loader that will render while the Assisted Insights process is in progress. This will automatically be added to the page when required and removed once the Assisted Insights data is returned. The default loader will be rendered if this is not provided.
- reportTitle {String} (Optional). The report title for the temporary Assisted Insights report. A default report title will be generated if this is not provided.

The options object also requires the following data for the Assisted Insights process. Aside from the required data, the variables will change depending on the type of charrt and analysis.

Required:

- type {String} - Indicates the type of Assisted Insights analysis. Either ‘compare’ or ‘explain’ (see below)

- metricField
- metricDescription
- categoryField
- value1 {String or Number}: The actual value used in the analysis

'explain' string type

Optional:

- dateValue
- timeSeries
- discreteTimeSeries
- granularity

'compare' string type

Additional required values

- categoryFieldId
- categoryKey
- value2 {String or Number}: The second value used in an Assisted Insights comparison
- valueOneFormatted (Only if using the default loader): Readable description of value one that will be displayed to the user (e.g. “Australia”)
- valueTwoFormatted (Only required if using the default loader): Readable description of value two that will be displayed to the user

Optional:

- timeSeries
- discreteTimeSeries
- timeGranularity

Example

See the assistedInsightsRequestedByChart event reference for how to call runAssistedInsights using data from the chart .

Calling it manually:

```
let assistedInsightsData = {
    categoryField: 60629,
    categoryFieldId: 1,
    categoryKey: "61039",
    dashboardName: "Sales Performance",
    dashboardUUID: "e7409ff2-f846-44e1-a603-b78ec51b20b9",
    metricDescription: "KPI",
    metricField: 60686,
    reportTitle: "Comparing KPI for Europe to Asia",
    parentReportId: 61035,
    sourceId: 54700,
    type: "compare",
    value1: "Europe",
    value2: "Asia",
    valueOneFormatted: "Europe",
    valueTwoFormatted: "Asia",
    viewId: 60543
};
reportAPI.runAssistedInsights(assistedInsightsData);
```

openComparePanel(value, comparableValues, metricDescription)

Returns

Nothing

Description

Renders a panel where the user can select a second value for use in the Assisted Insights comparison.

Users can implement their own compare panel by overwriting this function; otherwise, the default panel will be rendered. In both cases, the *compareValuesSelected* event should be triggered once the second value has been selected. See the *compareValuesSelected* event reference for information on what data should be passed with the event.

Calling *openComparePanel* is handled automatically by the default Assisted Insights flow if one value only is selected when the comparison is triggered from a chart tooltip. If *preventDefaultAssistedInsights* is true and you are calling *runAssistedInsights* yourself, *openComparePanel* will also need to be called manually

Parameters

- value {Object}: valueObject containing the first value in the comparison
  - These can be in either of the following forms, where value/rawValue refer to the value that will be used in the analysis and description /formattedValue refer to the value description that will be displayed to the user

```
{
   value: “EU”,
   description: “Europe”
}
```

```
{
   rawValue: "EU",
   formattedValue: "Europe"
}
```

> If you are extracting Assisted Insights data from the *assistedInsightsRequestedByChart* event, *value* corresponds to *value1* in the eventData and *comparableValues* corresponds to *possibleValues*

- comparableValues {Array}: Array of valueObjects containing the possible values that can be used as the second value in the comparison analysis
  - Must be in the form:

```
[{
   value: “AUS”,
   description: “Australia”
},
{
   value: “NA”,
   description: “North America”
}]
```

- metricDescription {String} (Optional): Description of the metric being used (eg, KPI). This is for UI purposes and is used only to add more information to the default compare panel.

Example

```
reportAPI.addListener('assistedInsightsRequestedByChart', event => {
let runAssistedInsightsData = event.eventData;
if (runAssistedInsightsData.type === 'compare' && runAssistedInsightsData.value2 == null) {
reportAPI.openComparePanel(runAssistedInsightsData.value1, runAssistedInsightsData.possibleValues, runAssistedInsightsData.metricDescription);
}
});
```

displayAssistedInsightsData(assistedInsightsData)

Returns

Nothing

Description

Renders the Assisted Insights data on the page.

Users can customize the way the data is rendered by overwriting this function. In order to do this, set *useDefaultAssistedInsightsPanel* to false on the Report API.

If you do render your own display panel, we recommend that you call *deleteTemporaryAssistedInsightsReport* once the data is no longer required (usually when the display panel is closed). This will remove any data related to the temporary assisted insights report. The default display panel already handles this automatically.

Example

```
let runAssistedInsightsPromise = reportAPI.runAssistedInsights(data);
runAssistedInsightsPromise.then(assistedInsightsData => {
// Display the report results
reportAPI.displayAssistedInsightsData(assistedInsightsData);
});
```

cancelAssistedInsights(assistedInsightsUUID)

Returns

Nothing

Description

Cancels the Assisted Insights process before it has completed and the data has been returned.

This can be useful when rendering a custom loader so users have the option to cancel the process if it is taking too long.

This requires the UUID of the Assisted Insights job. This is passed as eventData when the *assistedInsightsInProgress* event is triggered.

Example

```
reportAPI.addListener('assistedInsightsInProgress', event => {
let assistedInsightsTaskUUID = event.eventData;
reportAPI.cancelAssistedInsights(assistedInsightsTaskUUID);
});
```

deleteTemporaryAssistedInsightsReport(assistedInsightReportId)

Returns

Nothing

Description

Deletes the temporary report that is generated for Assisted Insights. This is automatically called when the default Yellowfin display panel is closed, but if you're rendering a custom display panel, we recommend calling it once the data is no longer required.

Parameter

- assistedInsightReportId {Number}: ID for the temporary Assisted Insights report. Returned in the data that is passed when the *runAssistedInsights* promise is resolved or when the *assistedInsightsCompleted* event is triggered.

Example

```
let runAssistedInsightsPromise = reportAPI.runAssistedInsights(data);

runAssistedInsightsPromise.then(assistedInsightsData => {

let reportId = assistedInsightsData.assistedInsightsReportId;

reportAPI.deleteTemporaryAssistedInsightsReport(reportId);

});
```

registerOutputType(outputOptions, callback)

Returns

Number

Description

Registers a report output type to the report, which allows developers to get extra information from a report. Returns a unique identifier for the output you have just registered, this can be used to remove the outputType request from the report when you no longer need it.

The callback function for the output type will be called whenever the report has returned its data from the server, but before the reportComplete event is triggered.

From Yellowfin version 9.2 onwards, you can request the reports dataset as an output type.

dataset

Returns the dataset for the report.

Options

No options can be defined for this output type

Callback Parameters

dataset

A two dimensional array. Each item in the array will be an Object containing the following values:

- **rawValue:** The value as it would be when retrieving it from the database.
    - **formattedValue:** The value after having any Ref Code or formatters applied to it.
    - **htmlFormattedValue:** The value ready to be inserted into an HTML page.

For the field “Gender” which by default is formatted as a Ref Code, the data object would look like the following.

```
{
	rawValue: 'FEMALE', //In the SkiTeam database the Gender code is all caps
	formattedValue: 'Female', //After it has been formatted as a refcode it becomes a more readable version
	htmlFormattedValue: 'Female' //For this case, they are exactly the same.
}
```

Differences between formattedValue and htmlFormattedValue

In a lot of cases you might not see any difference between the *htmlFormattedValue* and the *formattedValue*. There are some formatters which will output HTML tags which means that you can see differences there. In the following report example, the “Gender As Link” field is formatted using a “Link To URL” formatter which generates an anchor tag to be inserted into the table.

Gender	Gender As Link
Female	FEMALE

The dataset for the above row looks like this:

```
[{
	formattedValue: "Female"
	htmlFormattedValue: "Female"
	rawValue: "FEMALE"
},
{
	formattedValue: "FEMALE"
	htmlFormattedValue: "<a href="localhost:8080/myTestSite/FEMALE" target="_blank" rel="nofollow noopener noreferrer">FEMALE</a>"
	rawValue: "FEMALE"
}]
```

Where the *htmlFormattedValue* contains the actual anchor tag and the *formattedValue* is what is displayed to the user.

Parameters

outputOptions: String, Object

Object or String with the information about the output type you wish to register. If a string is passed, default options will be used for that particular output type.

callback: Function

Function to call when the output type has completed. The parameters objects passed to this callback will vary depending on the output type that has been registered.

Example

Register a dataset output type and output the dataset:

```
report.registerOutputType('dataset', function(reportDataset) {
	console.log(reportDataset);
});
```

Or register the dataset using the *outputInformation* object, these will achieve the same result:

```
report.registerOutputType({ resultType: 'dataset' }, function(reportDataset) {
	console.log(reportDataset);
});
```

removeOutputType(outputKey)

Returns

Nothing

Description

De-registers the report output type and callback associated with the passed *outputKey*. This will stop the Yellowfin server from returning the information for that output request.

Parameters

outputKey: Number

The *outputKey* that you wish to de-register.

Example

Register a dataset type and then remove the output type from the report after it has completed once.

```
let outputId = report.registerOutputType('dataset', function(data) {
        console.log(data);
       report.removeOutputType(outputId);
});
```

reset()

Returns

Nothing

Description

Clears all of the report interactions (drill, sorting, time sliders, etc.) from a report and then runs the report. This will not reset the filters associated with the report as it is possible that the filters object will be linked to multiple reports, so resetting a filter specifically for this report can have an unintentional effect on another report. If you want to completely reset the report use this function together with *filters.resetFiltersToDefault() or filters.clear()*.

Example

Add a reset button that can reset the reports interactions, but not filters:

```
let resetButton = document.querySelector('div#resetReportButton');
resetButton.addEventListener('click', function(e) {
       report.reset();
});

Add a reset button that can reset the reports interactions and its filters back to their default values

let resetButton = document.querySelector('div#resetReportButton');
resetButton.addEventListener('click', function(e) {
       report.reset();
       report.filters.resetFiltersToDefault();
});
```

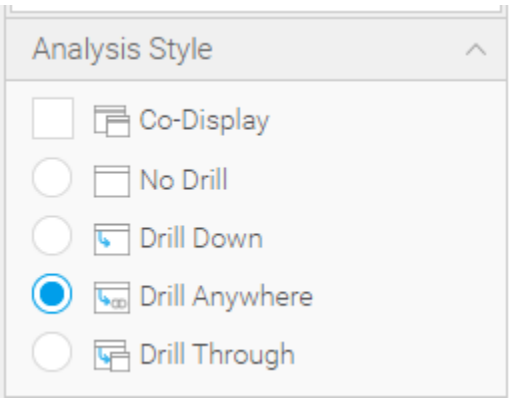
isDrillAnywhere()

Returns

Boolean

Description

Returns whether or not the report has been defined as a 'Drill Anywhere' type report. This will be true if the “Analysis Style” option is set to “Drill Anywhere” during report creation.



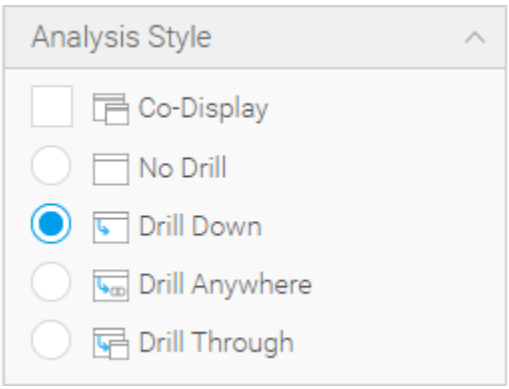
isDrillDown()

Returns

Boolean

Description

Returns whether or not the report has been defined as a 'Drill Down' type report. This will be true if the “Analysis Style” option is set to “Drill Down” during report creation.



isDrillDownApplied()

Returns

Boolean

Description

Returns whether or not the report has been drilled down. This will be true if any of the drill down fields on the report has been drilled down.

Example

```
console.log(report.isDrillDownApplied()); //Will return "false" as no drilling has been applied
report.drill(1, 'Agency');
console.log(report.isDrillDownApplied()); //Will now return "true" now that has drill down has been applied
```

isDrillAnywhereApplied()

Returns

Boolean

Description

Returns whether or not the report has had drill anywhere applied to it.

drill(fieldId, value, toField)

Returns

Nothing

Description

Applies drilling to the passed fieldId, with the passed value.

For 'Drill Anywhere' reports *toField* will be used to determine the field to replace the *fieldId* field with. For a 'Drill Down' report the *toField* will be ignored as there is an internal hierarchy of drilling already defined at the view level.

If the drill is successful a “drilldown” or “drillanywhere” event will be triggered.

Parameters

fieldId: Number, String

The field that you wish to drill on.

value: Number, String

The value you wish to apply as the drill value.

toField: Number

View Field Template ID of the field that you wish to drill to. Will only be used on Drill Anywhere reports.

Examples

Drill down on the value “Agency” on the “Agency Type” field using a fieldId:

```
report.drill(1, 'Agency');
```

Or we can also use the field name as the fieldId parameter:

```
report.drill('Agency Type', 'Agency');
```

drillUpLevels(fieldId, levels)

Returns

Nothing

Description

Drills up *levels* for the passed *fieldId*. If the field you are drilling is at the top of its hierarchy already, no further action will be taken.

Using fields Camp Region -> Camp Country -> Camp Name as an example. If we are drilled down to the Camp Name level and call *report.drillUpLevels (fieldId, 1);* it will be returned to the Camp Country level.

If we called *report.drillUpLevels(fieldId, 5);* which is greater than the total drill hierarchy length (which is 3), this call would effectively reset the drilling on that field and return the fields drill to the Camp Region level.

Parameters

fieldId: Number, String

The field that you wish to drill up on.

levels: Number

The number of levels to drill up.

Example

Drill up a single level on a fieldId 1:

```
report.drillUpLevels(1, 1);
```

drillUpOneLevel()

Returns

Nothing

Description

Drills up one level on all of the fields that are drillable in the report.

Example

Create a drill up button which drills the entire report up a level.

```
let drillUpButton = document.querySelector('div#drillUp');
drillUpButton.addEventListener('click', function() {
        report.drillUpOneLevel();
});
```

drillReset(fieldId)

Returns

Nothing

Description

Resets the drill on the passed *fieldId*. If no *fieldId is passed,* the entire report's drill state will be reset.

Parameters

fieldId: Number, String

The field that you wish to reset drill on.

Example

Reset the drill of the entire report:

```
report.drillReset();
```

Reset the drill of field 'Agency Region' by name:

```
report.drillReset('Agency Region');
```

Reset the drill of the field 'Agency Region' by fieldId:

```
report.drillReset(1);
```

timeslider(from, to, chartId)

Returns

Nothing

Description

Applies the *from* and *to* values to the passed chart's time slider. If no chartId is passed, then the report's default chart will be used. This will run the report to re-generate the chart with the applied time series values.

Parameters

from: Number

The time in milliseconds of the lower bound time slider value.

to: Number

The time in milliseconds of the upper bound time slider value.

chartId: Number, String

ChartId that you wish to apply the time slider values to.

Example

Set the time slider to show dates between 'August 2014' and 'November 2014':

```
//Create the date objects for the days we care about and get their time values
let fromDate = new Date('2014-08-01').getTime();
let toDate = new Date('2014-11-01').getTime();
report.timeslider(fromDate, toDate);

//Set the slider value based on the ChartUUID
let fromDate = new Date('2014-08-01').getTime();
let toDate = new Date('2014-11-01').getTime();
report.timeslider(fromDate, toDate, '0b808dd1-2114-42bc-a358-5fe7bf2ec052');
```

unitSelection(unit, chartId)

Returns

Nothing

Description

Applies time unit selection to the passed chart. If no chartId is passed, the report's default chart will be used instead. The chart needs to have visible unit selection turned on for the option to have any effect.

Parameters

unit: String

The time granularity you wish to apply to the chart. The possible values are:

- MILLISECOND
    - SECOND
    - MINUTE
    - HOUR
    - DAY
    - WEEK
    - MONTH
    - YEAR

Example

Apply unit selection of MONTH to the reports default chart

```
report.unitSelection('MONTH'); //No chart Id passed so the default chart will be used

let chartUUID = 'b779c293-a8ac-44cb-82f5-0c64da385333';


report.unitSelection('MONTH', chartUUID ); //Apply with a specific chart uuid

let chartName = 'Chart One';


report.unitSelection('MONTH', chartName); //Pass the chart name to determine which chart to use
```

convertDrillDownToArray() & converDrillAnywhereToArray()

Returns

Array[Object]

Description

Returns an Array of the currently applied drill state on the report.

Each object in the Array will contain the following information:

- **fieldId:** Number indicating which drill hierarchy we have drilled on.
    - **reportId:** The ID of the current report.
    - **reportKey:** The report's unique key.
    - **reportUUID:** The report's Publish UUID.
    - **templateId:** The view level TemplateId of the field we are currently on.
    - **value:** The value applied to this stage of the drill.

When iterating through the array, all drill objects on a field will be drilled in sequential order. This means that if you have two fields that you can drill down on and they are both drilled, the first N values in the Array will be related to the first field, where N is how far that field has been drilled. The rest of the array would be made up of drill values for the second field.

On a report with the following drill structure:

Agency Region -> Agency Country

Camp Region -> Camp Country -> Camp Name

Agency Region can drill down one level. Camp Region can drill down two levels. When they are both fully drilled down the array would contain three entries. The first would relate to "Agency Region", the second would related to "Camp Region" and the third would relate to "Camp Country" as that is the child of Camp Region.

top

Event Reference

All report events will be triggered with an object that contains the following properties:

- **eventData:** The data for the specific event. This will vary depending on the event that has been triggered. See each event to see what is included in the eventData object.
    - **metadata:** An Object that contains some basic data about the report the event was triggered on, including:
        - **reportId:** The Internal ReportId.
        - **reportUUID:** The report's UUID.
        - **reportKey:** A unique identifier for this instance of the report.
    - **eventName:** The name of the event that was actually triggered.

seriesSelection

Description

Triggered when series selection is applied to the report.

Parameters

eventData

- **Series:** (Array[Number])Array of the field IDs that were applied for this series selection event. This should only contain fields that are actually on the report.
    - **chartId:** The chartId the series selection was applied to.

sort

Description

Triggered when the report is sorted.

Parameters

eventData

- **fieldId:** The field that sorting was applied on.
    - **sortDirection:** The direction that the field was sorted. Will be “ASCENDING” or “DESCENDING”.

reportStart

Description

Triggered when the report sends a request to the server to get a new dataset.

Parameters

No Event Data.

reportComplete

Description

Called after the report has returned from the server, and all of the callbacks that were registered in the *registerOutputType* function are resolved.

Parameters

eventData will contain a number of output results from the report run.

Example of Event Data:

```
{ //Example Empty Dataset
        8399975157147:  [//A random ID for the output type
              [] //Report Dataset data
      ]
}
```

assistedInsightsRequestedByChart

Description

Triggered when a user prompts Assisted Insights from a chart tooltip.

Helpful if you want to extract the data used to generate a particular Assisted Insights report or if *useDefaultAssistedInsightsPanel* is false and you are calling *runAssistedInsights* yourself.

Parameters

The eventData object contains all of the information required to run Assisted Insights and can be passed directly to *runAssistedInsights* if you are calling it manually. The contents of this will change depending on the Assisted Insights type and the chart.

eventData

- categoryField
- categoryFieldId (optional)
- categoryKey (optional)
- dashboardName: Name of the dashboard
- dashboardUUID: UUID for the dashboard
- metricDescription
- metricField
- parentReportId: ID for the report that the chart belongs to
- possibleValues (optional): List of objects representing the values that can be used in a comparison analysis with value1
- reportKey: key for the report that the chart belongs to
- sourceId: ID for the Data Source used by the parentReport
- type: type of Assisted Insights analysis (either ‘explain’ or ‘compare’)
- value1: The value to be used in the analysis
- value2 (optional): The second value to be used in a comparison analysis
- viewId: ID for the view used by the parentReport

Example

```
reportAPI.addListener('assistedInsightsRequestedByChart', chartData => {

// This already contains all of the data we need to run Assisted Insights so we don’t need to make any additional changes to it

let newData = Object.assign({}, chartData.eventData);

let promise = this.reportAPI.runAssistedInsights(newData);

});
```

assistedInsightsStarted

Description

Triggered when *runAssistedInsights* is called.

Parameters

Nothing.

Example

```
reportAPI.addListener(assistedInsightsStarted, () => {

console.log(“Assisted Insights has started”);

};
```

compareValuesSelected

Description

Triggered when a second value is selected for comparison.

Parameters

eventData

- value1 {String or number}: The “actual” value one. This is what will be used in the analysis. Equivalent to the value/rawValue that is passed to *openComparePanel*
- valueOneFormatted {String or number}: The description of value one that will be displayed to the user. Equivalent to the description /formattedValue that is passed to *openComparePanel*
- value2 {String or number}: The “actual” value two. This is what will be used in the analysis. Equivalent to the value/rawValue that is passed to *openComparePanel*

- valueTwoFormatted {String or number}: The description of value two that will be displayed to the user. Equivalent to the description /formattedValue that is passed to *openComparePanel*

Example

```
this.reportAPI.addListener('compareValuesSelected', event => {

let comparisonValues = event.eventData;

// Combine them with the existing Assisted Insights data

let allData = Object.assign({}, assistedInsightsData, comparisonValues);

// Generate the assisted insights report

reportAPI.runAssistedInsights(allData);

});
```

assistedInsightsInProgress

Description

Triggered when the Assisted Insights background task is first started.

Parameters

eventData

- UUID of the Assisted Insights background task. This can be passed to *cancelAssistedInsights* when cancelling the task before it has completed

Note: The task UUID refers to the UUID for the background task and is different to the Assisted Insights report ID, which is returned when the *runAssistedIn sights* promise is resolved or the *assistedInsightsCompleted* event is triggered.

Example

```
reportAPI.addListener('assistedInsightsInProgress', event => {

let assistedInsightsTaskUUID = event.eventData;

reportAPI.cancelAssistedInsights(assistedInsightsTaskUUID);

});
```

assistedInsightUpdate

Description

Triggered while the Assisted Insights task is in progress. Provides updates on the current state of the task.

Parameters

eventData

- progressText {String}: A string containing translated updates on the current progress of the assisted insights task. Useful for rendering updates to the user on a custom loader
- state {String}: Current state of the background task. Can be Queued, Running or Complete

Example

```
reportAPI.addListener('assistedInsightUpdate', event => {

let progressData = event.eventData;

let customLoader = document.getElementById("myCustomLoader").

customLoader.innerHTML(progressData.progressText);

});
```

assistedInsightsCompleted

Description

Triggered when the entire *runAssistedInsights* process has finished.

Note: This is in addition to the *runAssistedInsights* promise being resolved. Both return the same data.

Parameters

eventData

- assistedInsightsReportId {Number}: ID for the temporary Assisted Insights report. This is required when calling *deleteTemporaryAssistedInsightsReport*
- chartData {Object}: Object containing Assisted Insights data objects to be rendered in the display panel
    - assistedInsightsImage {base64 String}: Image of the Assisted Insights chart
    - assistedInsightsZoomedImage {base64 String}: Larger version the image of the Assisted Insights chart
    - chartId {Number}: ID for the chart
    - narratives {Array}: Formatted text outlining the Assisted Insights findings

Example

```
reportAPI.addListener('assistedInsightsCompleted', data => {

// Display the report results

reportAPI.displayAssistedInsightsData(data.eventData);

});
```

assistedInsightsCancelled

Description

Triggered when *cancelAssistedInsights* has been cancelled.

Parameters

eventData

- UUID of the cancelled task

Example

```
reportAPI.addListener('assistedInsightsCancelled', event => {

console.log("Task " + event.eventData + " has been cancelled");

});
```

assistedInsightsError

Description

Triggered when something has prevented the Assisted Insights process from completing.

Parameters

eventData

- getMessageText {function}: Generic error message indicating an error has occurred

Example

```
reportAPI.addListener('assistedInsightsError', event => {
```

```
console.log(event.eventData.getMessageText());

});
```

assistedInsightsExceptionError

Description

Triggered when an exception occurs during the Assisted Insights process.

Parameters

eventData

- errorMessage: Generic error message indicating that the analysis could not be completed

Example

```
reportAPI.addListener('assistedInsightsExceptionError', event => {

console.log(event.eventData.errorMessage);

});
```

drilldown

Description

Triggered whenever drill down is applied to any field on the report.

Parameters

eventData includes:

- **fieldId:** The field that was drilled on.
    - **added:** (Array) Array of objects containing information about what values were applied
    - **drillState:** (Array) Array containing the entire drillState of the report. See *convertDrillDownToArray*

drillanywhere

Description

Triggered whenever 'Drill Anywhere' is applied to any field on the report.

Parameters

eventData includes:

- **added:** (Array) Array of drill objects that have been added in this drill event.
    - **drillState:** (Arra) Drill State of the entire report. See *convertDrillAnywhereToArray.*

drillUp

Description

Triggered whenever a report drills up. This will trigger for both 'Drill Anywhere' and 'Drill Down' reports.

Parameters

- **removed:** (Array) Array of drill objects that have been removed in this drill event.
    - **drillState:** (Array) Drill State of the entire report. See *convertDrillAnywhereToArray* or *convertDrillDownToArray.*

reportRunAlreadyInProgress

Description

Triggered if any process tries to start a report run while the report is already running.

Example

```
report.addEventListener("reportRunAlreadyInProgress", function() {

alert("report is already running");

});
```

resetDrill

Description

Triggered when the drill state for a field is reset or the entire reports drill state is reset. This will trigger for both Drill Anywhere and Drill Down reports.

Parameters

- **removed:** (Array) Array of drill objects that have been removed in this drill event.
    - **drillState:** (Array) Drill State of the entire report. See *convertDrillAnywhereToArray* or *convertDrillDownToArray*.

reset

Description

Triggered when the reports reset function is called.

Parameters

No Event Data.

slider

Description

Triggered when a chart's time slider value is modified.

Parameters

eventData

- **from:** (Number) Lower bound of the slider values.
    - **To:** (Number) Upper bound of the slider values.
    - **chartId: (**Number) The chart which the slider was modified on.

unitSelection

Description

Triggered when a chart's 'Unit Selection' is changed.

Parameters

eventData

- **unit:** (String) The unit to be applied to the chart.
    - **chartId:** (Number) The chart that the unit selection was applied to.

top