Dashboards API

How Dashboards Are Structured

A dashboard is a collection of one or more pages (known as Sub Tabs). Sub Tabs are a collection of Entities. An entity in the context of a dashboard, is anything that is placed onto a Sub Tab. This could be a report or a filter list as well as any object that can be placed onto a canvas (Shapes, Text, Buttons, Images, etc).

Thus, the basic structure of a dashboard looks like this:

- Dashboard
  - Sub Tabs
    - SubTab Entities (Reports, Canvas, etc.)
  - Global Content Entities (Filter Panels)
  - Filters API

Sub Tabs, Reports and Filters all have a number of their own properties that can be seen in the reference guides for those areas.

What is an Entity

As mentioned, an entity in the context of a dashboard is anything that is placed onto a Sub Tab. This could be a report or a filter list as well as any object that can be placed onto the canvas (Shapes, Text etc).

During Dashboard creation, when an entity is added to the Dashboard it is assigned a unique identifier. This is the identifier that is used for that Dashboard Entity internally within the DashboardAPI.

For reports, this unique ID is very important as it allows us to include the same report more than once on a dashboard and apply different filters to it. Otherwise if we had 2 versions of the same report we would not know which one to refer to when applying filters or drilling into that report.

Filter Panels and FiltersAPI

The FiltersAPI on a dashboard contains every single user prompt filter that has been enabled on that dashboard. Even if they are filters that aren't currently visible.

A Filter Panel is what the end-user sees. This uses the FilterObjects within the FilterAPI to generate what is displayed to the user.

It is not possible to manipulate Filter Panels without using DOM manipulation. However developers can manipulate the filters within those panels through the FiltersAPI.

Report References

You can see all of the reports and their entity IDs by using the **getAllReports()** function. This will return an Array of Objects which contains:

- **reportUUID -** This is the publishUUID of the report. This may not be unique for a Dashboard if the same report is added more than one time.
- **entityUUID -** This is the unique ID for this item on the dashboard.
- **subTabUUID -** The UUID of the Sub Tab that the report has been added to.
- **subTabId -** The internal ID of the Sub Tab that the report has been added to.

All of these reports will be automatically loaded by the dashboard when the sub tab becomes visible.

Dashboards and Presentations

Yellowfin Present is built off of the Yellowfin Dashboard. As such all of the functions within the DashboardAPI will work when you apply them to a Presentation. If you wish to navigate to the next slide when using a Presentation you can simply call the “nextSubTab()” function, or if you wish to move to the fifth slide, moveToTabAtPosition(4).

Property Reference

uuid

Returns

String

Description

Returns the UUID of the dashboard.

filters

Returns

FiltersAPI

Description

Returns the FiltersAPI for the Dashboard.

name

Returns

String

Description

Returns the dashboard name. This will be translated into the users language if translations are enabled and a translation has been provided for that language.

description

Returns

String

Description

Returns the dashboard description. This will be translated into the users language if translations are enabled and a translation has been provided for that language.

currentSubTabId

Returns

Number

Description

Returns the sub tab ID of the sub tab the user is currently viewing.

runUrl

Returns

String

Description

Returns a direct link to this dashboard. For example,

```
http://localhost:8080/RunDashboard.i4?dashUUID=1e68d9cc-fa5a-44e2-816d-782aa40ceeae&primaryOrg=1&clientOrg=1
```

This is a simple link to access this dashboard directly, it doesn't contain any filters.

externalApiUrl

Returns a direct embed link for this dashboard. For example,

```
<script type="text/javascript" src="http://localhost:8080/JsAPI/v3?dashUUID=1e68d9cc-fa5a-44e2-816d-782aa40ceeae"></script>
```

Function Reference

loadReport(options)

Returns

Promise

Description

Loads a report within the context of the dashboard. Resolves the promise and passes a Report object as the parameter, but does not automatically display it.

If the reportUUID/entityUUID combination has already been loaded for this dashboard, the promise will resolve immediately with the relevant Report object.

If the report has been set up as part of this dashboard and had filter linking and interaction linking applied, it will automatically have those links applied to it.

If the report is unrelated to this dashboard it will be loaded but will not be able to automatically use filters.

Parameters

Options - Object

Must contain the following:

- **ReportId** - (String) The ReportUUID or PublishUUID of the report you wish to load.
- **EntityUUID** - The unique ID for this instance of the report.

getCurrentSubTab()

Returns

SubTab

Description

Returns the SubTab object for the currently selected tab. If the dashboard has no currently selected tab (this can happen in some cases when it first loads) the first sub tab will be returned.

Example

```
let subTab = dashboard.getCurrentSubTab();
console.log(subTab.name);
```

getSubTabByPosition(position)

Returns

SubTab

Description

Returns the SubTab object for the passed *position*. This position starts at 0.

If a *position* value is outside of the number of tabs, null will be returned.

Parameters

position: Number

Example

Get the 4th sub tab and verify it exists and then output its name.

```
let subTab = getSubTabByPosition(4);
if(subTab) {
        console.log(subTab.name);
}
```

getSubTab(subTabId)

Returns

SubTab

Description

Returns the SubTab object for the passed *subTabId*. If a SubTab name is passed in, it will attempt to be converted to its corresponding tab ID.

If no matches can be found, null will be returned.

Parameters

subTabId: String

Examples

```
//Get sub tab by Id
let subTab = dashboard.getSubTab(123456);
//Get subTab by name
let subTab = dashboard.getSubTab('Campaign Summary');
```

getSubTabFilters(SubTab)

Returns

Object - {String, FilterObject}

Description

Returns an Object that contains all of the user prompt filters for the SubTab that is passed to the function. This object is keyed by Filter UUID.

Notes/Limitations

This function is useful for observing which filters are available for a given SubTab. If you are looking for a specific filter it is recommended using *filters.getFilter(filterId)* rather than getAllFilters()[uuid] as *getFilters* can accept a UUID or a name. If you wish to iterate over all filters, we recommend using *filters.forEach(fn)* instead.

nextSubTab()

Returns

Nothing

Description

Changes the Dashboard's current tab to the next one by order. From a user perspective this would move the tab to the right of the current tab.

If the dashboard is currently on the last tab, this will move the dashboard to the first tab.

Example

Move the user to a new tab after they click the "next" button:

```
document.querySelector('div#nextTabButton').addEventListener('click', function(e) {
        dashboard.nextSubTab();
});
```

Move the user to a new tab every 10 seconds:

```
setInterval(function() {
        dashboard.nextSubTab();
}, 10000);
```

previousSubTab()

Returns

Nothing

Description

Changes the Dashboard's current tab to the previous one by order. From a user perspective this would move the tab to the left of the current tab.

If the dashboard is currently on the first tab, this will move the dashboard to the last tab.

Example

Move the user to a new tab after they click the "next" button:

```
document.querySelector('div#previousTabButton').addEventListener('click', function(e) {
        dashboard.previousSubTab();
});
```

Move the user to a new tab every 10 seconds:

```
setInterval(function() {
        dashboard.previousSubTab();
}, 10000);
```

findSubTabOrder(subTabId)

Returns

Number

Description

Returns the position of the sub tab. If there is no matching sub tab, null will be returned. Sort indexing starts at 0.

Parameters

subTabId: Number, String

ID or name of the sub tab you wish to find the position of.

Examples

Get the sub tab position by name:

```
let subTabPosition = dashboard.findSubTabOrder('Campaign Summary')
```

Get the sub tab position by ID:

```
let subTabPosition = dashboard.findSubTabOrder(123456);
```

moveToTab(subTabId)

Returns

Nothing

Description

Changes the dashboard's current sub tab to the sub tab that matches the passed ID. If a name is passed, it will attempt to convert that to a matching ID. If there is no matching tab, then nothing will happen.

If the tab changes, a tabChanged event will be fired by the DashboardAPI.

Parameters

subTabId: String

UUID or Name of the sub tab you wish to navigate to.

Examples

Move to the tab ‘Campaign Summary’:

```
dashboard.moveToTab('Campaign Summary');
```

Move to the tab with uuid “28c595a5-c097-445c-ac9d-cbd4c415a667”:

```
dashboard.moveToTab(‘28c595a5-c097-445c-ac9d-cbd4c415a667’);
```

moveToTabAtPosition(position)

Returns

Nothing

Description

Change the dashboard's current sub tab to the sub tab at the matching *position*. If there is no matching tab, then nothing will happen. If the subTabPosition isn’t within the possible range (i.e. the number of sub tabs on the dashboard), nothing will happen.

If the tab changes, a tabChanged event will be fired by the DashboardAPI.

Parameters

position: Number

The sort order of the tab you wish to navigate to. Sort index starts at 0.

Examples

Move to the first sub tab:

```
dashboard.moveToTabAtPosition(0);
```

Move to the last sub tab:

```
dashboard.moveToTabAtPosition(dashboard.getSubtabs().length - 1);
```

hasFilters()

Returns

Boolean

Description

Returns true if the dashboard has any user prompt filters enabled. Otherwise returns false.

Example

```
let hasFilters = dashboard.hasFilters();
if(hasFilters) {
	console.log("this dashboard has modifiable filters");
}
```

refresh()

Returns

Nothing

Description

Re-runs all of the reports on the dashboard. This will force all of the reports to fetch new datasets from the server.

This can be useful in circumstances where you are working on live data and an action from the page will update the datasets behind some of the reports. This can then be used to re-run the reports with the same filter sets.

Example

```
dashboard.refresh();
```

resetDashboard()

Returns

Nothing

Description

Resets all of the reports on the dashboards to their default state, as well as resetting all of the filters to their default value.

This will reset all of the brushing and drill states on the reports as well as any linking of those filters.

Only reports that are on the current tab will be re-run.

goToReport(reportUUID)

Returns

Nothing

Description

Navigates the browser to the report output page associated with the passed reportUUID. If the report can’t be found then the user will be shown a “Report Can’t be Found page” message.

Parameters

reportUUID: String

PublishUUID of the report you wish to navigate to.

Limitations

This will only function when being used within Dashboard Code Mode.

getEmbedURL()

Returns

String

Description

Gets the embed URL for the dashboard exactly as it stands now. The URL will include any applied filters or bookmarks.

This differs from the property *externalApiUrl* as that only includes the link to the dashboard in its base state (i.e. without any filter or bookmark references).

getSubTabs()

Returns

Array[SubTab]

Description

Returns all of the sub tabs associated with the Dashboard in an ordered array.

Example

Print all of the sub tab names and UUIDs in order:

```
let subTabs = dashboard.getSubTabs();
subTabs.forEach(subTab => {
        console.log(subTab.name + " : " + subTab.uuid);
});
```

getAllReports()

Returns

Array[Object]

Description

Returns an array of objects containing a Report's UUID, the EntityUUID it has been assigned on the dashboard, as well as the sub tab it has been placed on.

If there are no reports on the dashboard, an empty array will be returned.

Example

This function can be used to search for all instances of a particular report that might be on the dashboard:

```
let reportToFind = 'af67e527-81d3-47fc-81ce-dfc506a61dd2';
let dashboardReports = dashboard.getAllReports();
let dashboardReport= dashboardReports.find(reportInfo => {
        return reportInfo.reportUUID ===  reportToFind;
});

if(dashboardReport) {
        //We found that report now we can load the report object. If the report has already been loaded by the dashboard this will just give us that version
of the report
        dashboard.loadReport({
                reportId: reportInfo.reportUUID,
                entityUUID: reportInfo.entityUUID
        }).then(report => {
                //Now we've fetched the report.
        });
}
```

forEach(fn)

Returns

Nothing

Description

Iterates over all of the sub tabs that are included on the dashboard in order. The function will be passed a SubTab object on each iteration.

Example

Print the name and UUID of all of the sub tabs on the dashboard:

```
dashboard.forEach(subTab => {
	console.log(subTab.name + " : " + subTab.uuid);
});
```

Event Reference

tabChanged

Description

Triggered whenever the current sub tab or slide is changed.

Parameters

Event

Contains:

- **currentSubTab**: (SubTab) This is the tab that we have just moved to.
- **currentSubTabId**: (Number) The ID of the sub tab we are now on.
- **previousSubTabId**: (Number) - The ID of the sub tab we just left.

Examples:

```
dashboard.addEventListener('tabChanged', function(event) {
	let newTab = event.currentSubTab;
	console.log('We have just moved to the ' + newTab.name + ' tab');
});
```

```
dashboard.addEventListener('tabChanged', function(event) {
	let lastTab = dashboards.getSubTab(event.previousSubTabId);
	console.log('Moved from ' + lastTab.name);
})
```