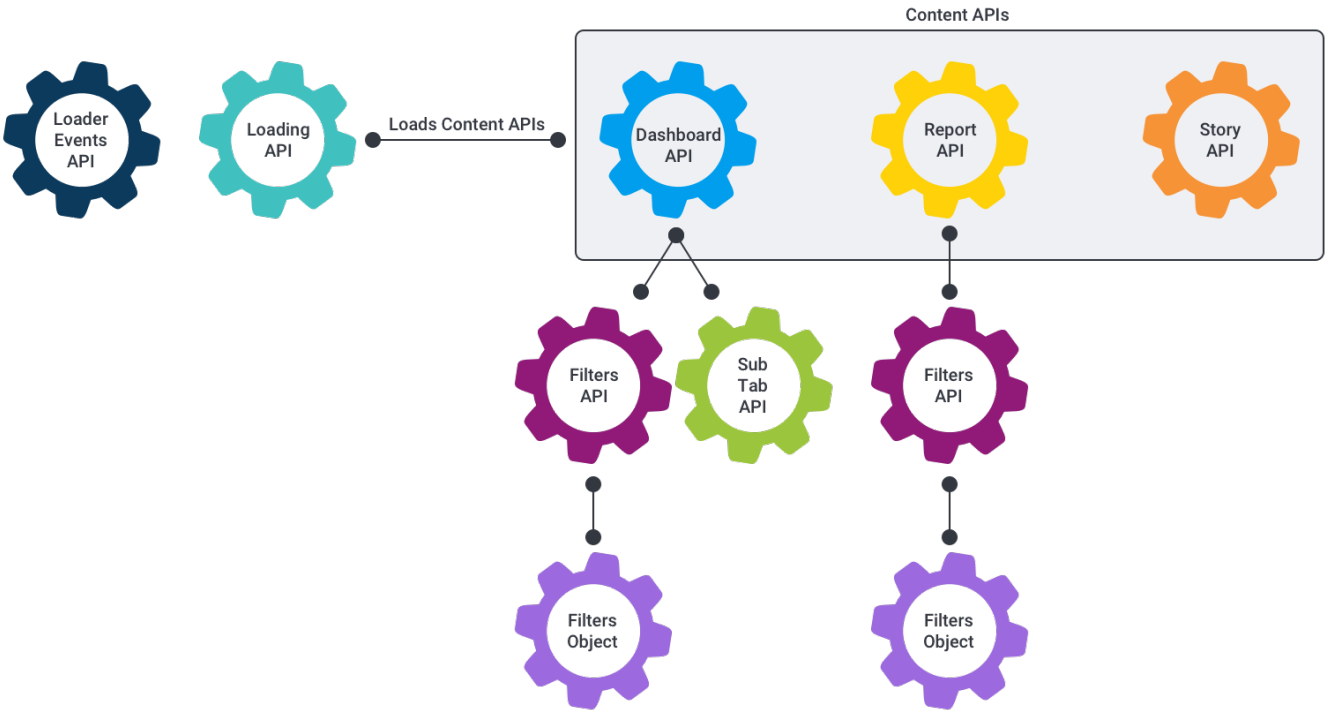
JS API Components

What is the Yellowfin JavaScript API?

The JavaScript API (JS API) is a tool that allows third party developers to embed and manipulate Yellowfin content. Generally when someone refers to the JS API they are referring to the embedding functionality.

The JS API is made up of a lot of parts. There are two broad API types that objects fall into when talking about the JS API — Loading APIs and Content APIs.



Loading APIs

These APIs essentially allow you to load content, such as a report, dashboard or story. They don't provide too much functionality except for loading the Content API for the object you asked for.

yellowfin.reports.loadReport({ reportUUID : 'auuid' })

As of 9.6, loading APIs are almost exclusively available through the external JS AP only. These are the dashboard, story and report loading APIs. They all provide a single loadContent (loadReport, loadDashboard and loadStory) function.

All of these functions execute asynchronously and return a promise which will resolve with the loaded Content API after the content has loaded.

Content APIs

Content APIs are the actual content — and they cover a much broader range of types. Currently there are three top-level Content APIs which are Story, Dashboard and Report. However they all contain child objects that make them much more complex.

Both Report and Dashboard APIs will contain a Filters API if the content has been configured to have filters. This allows developers to get access to filter objects and manipulate them via code, or react to their events.

Report API

The Report API provides access, via code, to most of the functionality that a user can perform on a report. Nearly all user interactions that occur on a report are now directed through the Report API — with the exception of table controls (navigating between pages and sections) and drill through.

This means that a user should be able to perform via code nearly any action they can perform in the UI, including applying time slider values, unit selection, series selection, sorting, drill down and drill anywhere. All of these interactions also have an event associated with them, so developers can listen for these events occurring. They will be triggered whenever the event occurs (whether via a code call or UI interaction).

Ideally, our code functionality should fully match our UI — and future interactions that we implement should be implemented via the API first, and then have the UI built around that.

The Report API will listen to its Filters API for any changes to the filters applied values; if they occur, the API will trigger a re-run of the report. This trigger incorporates a short delay (100 milliseconds) before re-run so that multiple changes can be made to a report simultaneously.

Output types

The Report API also provides the ability to retrieve an “output type” for a report, using the registerOutputType functionality. The registerOutputType adds a request to a report run which will say “also return this information” and provides a callback to be passed in. This means that if the report’s filters are changed and a new run is triggered, the passed callback will be completed when the new result is completed.

This allows developers to retrieve report datasets, unique values for columns and may allow other functionality to be returned in coming releases.

Dataset

The dataset output type returns the entire dataset for the report. This will be in a particular data order and only contain data. The Report API provides a function (getFieldsInDatasetOrder()) which will return all of the reports fields in dataset order, which means you can match a field to its data based on its array position.

uniqueValues

The *uniqueValues* output type returns all of the distinct values for any passed fields. This is particularly useful for developers who wish to fetch all of the values for a particular dimension that would return a result if they were applied as a filter.

Display

The Report API also provides functionality to create visualisation elements based on the report.

The function *report.createReportElement(options);* creates a visualisation of any chart, table or canvas that is present on that report. It displays them simultaneously (so Chart A and Chart B can sit next to each other).

Dashboard API

A Dashboard is built upon many subtypes — sub tabs, reports, filters, canvases and widgets.

A dashboard is a collection of sub tabs. Each of these sub tabs contains either:

- a canvas, which allows placing of many reports and widgets; or,
- a grid layout which has reports, filters and certain widgets.

The Dashboard API helps to keep track of what page the user is currently on and provides a Filters API which has the state of any filters on the dashboard. All the reports that are loaded on a dashboard listen to this Filters API, so provided the filters are linked, the Report API will just listen for its filter changes, just as it would when embedding a report without a dashboard.

Sub Tab API

The Sub Tab API provides metadata about the passed dashboard tab. This API doesn’t provide many functions that can be called, but it can find all the reports on a sub tab and retrieve the Canvas API if there is a canvas.

Filters API

The Filters API provides a representation of the filters on a particular piece of content. It provides functionality to set or retrieve filter values, as well as updating the possible values that can be used for cached reports.

The API contains a set of FilterObjects, which are the API objects for each individual filter that is available on the piece of content.

It also provides functionality to create filter list elements. This lets you place the Yellowfin filter lists anywhere on a page and offers the option of customising which filters will be displayed within that filter list.

The Filters API also lets you listen for any change in filters across the entire Filters API, so you can react to any filter on the content changing.

The Filter object

This object is a representation of an individual filter, within the Filters API. It's used to maintain the state of the filters and trigger events when a filter value changes.

A filter has two sets of values, described in the following table.

Value type	Description
Applied values	The values applied to the report or dashboard, used in any queries that Yellowfin generates when running the report.
Staged values	The current values visible to the user in the report or dashboard. Once an apply() function is called on the Filter Object or Filters API, these values will be copied to the applied values.

The Filter Object also provides functionality to create an individual display representation of the filter itself. This uses the same styling that a normal Yellowfin filter would, but it is completely independent of a filter list.

Loader events

The Loader Events API lets you listen for standard Yellowfin loading events and react to those events.

This allows you to prevent the loader from being added, or to attach it to a different element when detected. For example, if you have a canvas with many reports, instead of attaching a loader to each report on the canvas, you could attach it to the overall canvas object and then remove the loader when all of the reports have finished rendering.

JS API Components#top

Internal vs external JS API

What does this mean?

When using the term "Internal JS API" this refers to the tools used by

- the Yellowfin development team to create web app features; or,
- a third-party developer using Yellowfin code mode or a custom header.

The term "External JS API" is used to reference a developer embedding Yellowfin content into an external page.

There are effectively two Yellowfin JS APIs.

The external JS API is our offering for developers who embed Yellowfin content into an external page. We provide details of the external JS API on our external wiki.

The internal JS API is used by the Yellowfin development team. Third-party developers unknowingly use this version of the JS API when accessing code via Yellowfin code mode or a custom header.

Both JS APIs are explained in more detail below.

External JS API

When using the JS API externally it gives you access to all of the Yellowfin Loading APIs and Content APIs. Which means you can load any piece of content you wish, provided you know the content's UUID.

In most cases a report or dashboard listing service can be created using Yellowfin webservices to fetch YF content objects, and then use the UUIDs provided from those objects to load the object in the JS API. This makes it possible to build a completely custom UI while using Yellowfin reports, dashboards and stories within them, so the content building can be done by non-developers.

Considerations

Authentication can sometimes be problematic because the Yellowfin server is treated as a third-party application. Browsers that reject third-party cookies on a site therefore reject the Yellowfin JS API (Safari and Chrome in Incognito mode). In addition, security changes around the “Same Site” Flag makes it difficult to configure a web server to accept all the cookies that are required.

The other issue is differences in how functionality works. The following table provides a list of features that work differently or are not yet enabled within the external JS API.

Functionality	Internal JS API behaviour	External JS API behaviour
Dashboard & Presentation toolbars	Create broadcast.	–
Dashboard & Presentation toolbars	Share and see users who have favourited content in the toolbar which appears in the top right of these pieces of content).	–
Report drill through (when not using the drill through pop up)	Drill through will redirect the user to the report output page.	Drill through will simply replace the parent report with the child report.
Open report from dashboard	It’s possible to open a report from an embedded dashboard.	–
Embed signal widgets on dashboard	Signal widgets can be embedded — with links that work.	Signal widgets can be embedded, but any links within them won’t work.
Embed story widgets on dashboard API	Story widgets can be embedded — with links that work.	Story widgets can be embedded, but any links within them won’t work.

Internal JS API

The Yellowfin JS API currently has no access to any of the Loading APIs that are present within the external JS API. This means that for any piece of content to be loaded it must be done via a Yellowfin action. Which means that third-party developers don’t have as much control over how they could display reports or other content. As it all needs to be loaded (and in most cases) rendered, by Yellowfin first.

All Yellowfin content and its individual features are generally available to the Content APIs and the content being rendered internally. As the way we generally build the functionality internally first, and then attempt to get it to work externally.

Considerations

Content APIs are only available via Code Mode. There are no loading APIs.

This makes it challenging to create a customized app experience. Code Mode lets you customize a dashboard heavily, but you can’t load extra dashboards from within that dashboard, nor reports that haven’t been added.

JS API Components#top