Advanced Use

Overview

top
If you want to have more control over the loading of content, call reports or dashboards on demand, or set display options dynamically (based on user input), you can call the API directly from your own script.

The Javascript API must be included before any API calls can be made:

```
<script src="http://localhost/JsAPI" type="text/javascript"></script>
```

A specific version of the API may be requested using the `version` parameter:

```
<script src="http://localhost/JsAPI?version=2.1" type="text/javascript"></script>
```

If the browser is unable to load the API, any calls to load reports or dashboards will fail. If you wish to detect whether the API has loaded successfully, you should check the variable `window.yellowfin` is available:

```
<script src="http://localhost/JsAPI" type="text/javascript"></script>
<script type="text/javascript">
if (!window.yellowfin) {
   alert('Error loading API');
}
</script>
```

Server Information

top
After loading the API, some server information is made available:

	Description
`yellowfin.apiVersion`	The version of the API being used by the server.
`yellowfin.baseURL`	The base URL used to connect to the API on the server
`yellowfin.serverInfo.releaseVersion`	The release version of Yellowfin running on the server (eg. "6.1")
`yellowfin.serverInfo.buildVersion`	The build version of Yellowfin running on the server (eg. "20120601")
`yellowfin.serverInfo.javaVersion`	The java version installed on the server
`yellowfin.serverInfo.operatingSystem`	The Operating System running on the server
`yellowfin.serverInfo.operatingSystemArch`	The Operating System architecture on the server
`yellowfin.serverInfo.operatingSystemVersion`	The Operating System version on the server
`yellowfin.serverInfo.schemaVersion`	The schema version of the Yellowfin configuration database

Example

```
<script src="http://localhost/JsAPI" type="text/javascript"></script>
<script type="text/javascript">
if (window.yellowfin) {
   alert('Yellowfin API loaded. Version: ' + yellowfin.apiVersion);
}
</script>
```

Loading a Report

top

A report is loaded by calling the `yellowfin.loadReport` function:

```
yellowfin.loadReport(options);
```

Options are passed to the function as a Javascript object. These include a report identifier for the report you are loading, the elementId of the HTML element in which to load the report (or the element itself), and other options that alter the way the report is displayed. The available options are:

Option	Description
`reportUUID`	Either `reportUUID`, `reportId` or `wsName` must be present. The unique ID identifying the dashboard to load.
`reportId`	Either `reportUUID`, `reportId` or `wsName` must be present. The numeric reportId identifying the report to load. It is recommended to use the `reportUUID` parameter instead.
`wsName`	Either `reportUUID`, `reportId` or `wsName` must be present. The Web Service name identifying the report to load. It is recommended to use the `reportUUID` parameter instead.
`elementId`	Either `elementId` or `element` must be present. The id of the html element in which to load the report.
`element`	Either `elementId` or `element` must be present. The html element in which to load the report.
`showTitle`	Default: `true` Set to `false` to omit the title bar at the top of the report. All interactive buttons included in the title bar will also be omitted.
`showInfo`	Default: `true` Set to `false` to omit the Info button in the title bar.
`showFilters`	Default: `true` Set to `false` to omit the Filters button in the title bar. Any user-prompt filters will not be displayed.
`showSections`	Default: `true` Set to `false` to omit the Sections button in the title bar (for reports with tabbed or multi-page sections).
`showSeries`	Default: `true` Set to `false` to omit the Series button in the title bar (for reports with the series selection option).
`showPageLinks`	Default: `true` Set to `false` to omit the previous page/next page button in the title bar (for reports with multiple pages).
`showExport`	Default: `true` Set to `false` to omit the Export button in the title bar.
`height`	Default: automatically detected from the dimensions of the enclosing element Set this to a numeric value to override the report height.
`width`	Default: automatically detected from the dimensions of the enclosing element Set this to a numeric value to override the report width.
`display`	Default: `chart` Set to `table` to display the report initially as a table. Set to `chart` to display the report initially as a chart. This is ignored for reports that do not have both table and chart available.
`fitTableWidth`	Default: `true` Set to `true` to attempt to scale the report to the width of the enclosing element.
`canChangeDisplay`	Default: `true` Set to `false` to omit the buttons that allow the user to switch between chart and table display.

`filters`	Set to an object containing filter values to pass to the report.
`username`	Set this along with the `password` parameter to authenticate as a particular user when loading the report. This avoids the need for users to enter their login details before viewing restricted reports.
`password`	Set this along with the `username` parameter to authenticate as a particular user when loading the report.

Examples

This example loads a report into an element specified by its universal id, setting some initial display options:

```
var options = {};
options.reportUUID = 'e5e5aaf3-c3b8-4f9b-8280-e21e4d848e63';
options.elementId = 'myReport';
options.showFilters = 'false';
options.showSeries = 'false';
options.display = 'chart';
options.fitTableWidth = 'false';
yellowfin.loadReport(options);
```

This example does the same thing with an anonymous options object:

```
yellowfin.loadReport({
   reportUUID: 'e5e5aaf3-c3b8-4f9b-8280-e21e4d848e63',
   elementId: 'myReport',
   showFilters: 'false',
   showSeries: 'false',
   display: 'chart',
   fitTableWidth: 'false'
});
```

This example passes the element directly rather than just its id:

```
yellowfin.loadReport({
   reportUUID: 'e5e5aaf3-c3b8-4f9b-8280-e21e4d848e63',
   element: document.getElementById('myReport')
});
```

Loading Report Filters

top

Filters used by a report can be loaded by calling the `yellowfin.reports.loadReportFilters` function. To use this function, load the reports sub-API into your page along with the main API:

```
<script src="http://localhost/JsAPI" type="text/javascript"></script>
<script src="http://localhost/JsAPI?api=reports" type="text/javascript"></script>
```

Then call the `loadReportFilters` function:

```
yellowfin.reports.loadReportFilters(reportId, callback, arg);
```

The first argument is the unique identifier for the report, which may either be a `reportUUID` or a `reportId`. We recommend using the `reportUUID` where possible. The second argument is a callback function that will be called by the API when the filters for the report have been loaded. The first argument to the callback function will be the list of filters in the report. The second argument to the callback function will be the third argument supplied to the `loadReportFilters` function (if specified).

The filters object returned as the first argument to the callback function is an array containing any filters used in the report. Each element in the array is an object containing information about that filter. These filter objects contain the properties:

Property	Description
`filterUUID`	A unique identifier for the filter.

`filterId`	A numeric identifier for the filter.
`nativeType`	The native data type of the filter.
`description`	The description of the filter.
`operator`	The operator used with the filter.
`display`	The display style used by the filter.
`dependencies`	Set to true if other filters in the report are dependent on this one.
`list`	Set to true if the filter is a list style (allows multiple values).
`between`	Set to true if the filter is a between style (requires a start and end value).
`listValues`	If the filter is displayed as a drop-down list, this property contains a list of available options.

Examples

This example loads the report filters and displayed them to the user:

```
function filterCallback(filters) {

   for (var i = 0; i < filters.length; i++) {
      alert('Filter ' + filters[i].description + ' (' +
         filters[i].filterUUID + '), display style: ' +
         filters[i].display);
   }

}

yellowfin.reports.loadReportFilters(
   'e5e5aaf3-c3b8-4f9b-8280-e21e4d848e63', filterCallback);
```

This function can be used to load the available filters, and then pass them back to the `loadReport` function to set up initial filter values for the report when it is loaded into the page. For example:

```
function filterCallback(filters) {

   var filterValues = {};

   for (var i = 0; i < filters.length; i++) {

      if (filters[i].description == 'Country') {

         filterValues[filters[i].filterUUID] = 'Australia';

      } else if (filters[i].description == 'Start Date') {

         filterValues[filters[i].filterUUID] = '2011-01-01';

      } else if (filters[i].description == 'Invoiced Amount') {

         filterValues[filters[i].filterUUID] = 6400;

      }

   }

   // set up other options to load the report
   var options = {};
   options.reportUUID = 'e5e5aaf3-c3b8-4f9b-8280-e21e4d848e63';
   options.elementId = 'myReport';
   // add the filter values
   options.filters = filterValues;

   // load the report
   yellowfin.loadReport(options);

}

yellowfin.reports.loadReportFilters(
   'e5e5aaf3-c3b8-4f9b-8280-e21e4d848e63', filterCallback);
```

Filter values passed to the `loadReport` function should be specified as simple values as above. If the filter is a list style, multiple values can be set using an array:

```
filterValues[filterUUID] = ['Australia', 'China', 'Italy'];
```

If the filter is a between style, the start and end values should be set using an array:

```
filterValues[filterUUID] = [500, 600];
```

The `options.filters` element passed to the `loadReport` function should contain values keyed either by `filterUUID` or `filterId`. We recommend using `filterUUID` where possible.

Loading a Dashboard

top
A dashboard is loaded by calling the `yellowfin.loadDash` function:

```
yellowfin.loadDash(options);
```

Options are passed to the function as a Javascript object. These include an identifier for the dashboard you are loading, the elementId of the HTML element in which to load the dashboard (or the element itself), and other options that alter the way the dashboard is displayed. The available options are:

Option	Description
`dashUUID`	Must be present. The unique identifier for the dashboard to load.

`elementId`	Either `elementId` or `element` must be present. The id of the html element in which to load the dashboard.
`element`	Either `elementId` or `element` must be present. The html element in which to load the dashboard.
`showTitle`	Default: `true` Set to `false` to omit the title bar at the top of the dashboard. All interactive buttons included in the title bar will also be omitted.
`showInfo`	Default: `true` Set to `false` to omit the Info button in the title bar.
`showFilters`	Default: `true` Set to `false` to omit the Filters button in the title bar. Any analytical filters will not be displayed.
`showExport`	Default: `true` Set to `false` to omit the Export button in the title bar.
`height`	Default: automatically set from the dimensions of the reports in the dashboard. Set this to a numeric value to override the dashboard height. If the reports in the dashboard require more space, a vertical scrollbar will be added.
`width`	Default: automatically set from the logged-in user's preferences or the system configuration setting Set this to a numeric value to override the dashboard width. Set this to `auto` to use the full width of the enclosing element.
`filters`	Set to an object containing filter values to pass to the dashboard.
`username`	Set this along with the `password` parameter to authenticate as a particular user when loading the dashboard. This avoids the need for users to enter their login details before viewing restricted dashboards.
`password`	Set this along with the `username` parameter to authenticate as a particular user when loading the dashboard.

Examples

This example loads a dashboard into an element specified by its id, setting some initial display options.

```
var options = {};
options.dashUUID = '3b0b6c9a-9dfb-41f0-b85a-eb17bb8aeeb9';
options.elementId = 'myDash';
options.showFilters = 'false';
options.showExport = 'false';
yellowfin.loadDash(options);
```

This example does the same thing with an anonymous options object:

```
yellowfin.loadDash({
   dashUUID: '3b0b6c9a-9dfb-41f0-b85a-eb17bb8aeeb9',
   elementId: 'myDash',
   showFilters: 'false',
   showExport: 'false'
});
```

This example passes the element directly, rather than just its id:

```
yellowfin.loadDash({
   dashUUID: '3b0b6c9a-9dfb-41f0-b85a-eb17bb8aeeb9',
   element: document.getElementById('myDash')
});
```

Loading Dashboard Filters

top

Filters used by a dashboard can be loaded by calling the `yellowfin.dash.loadDashFilters` function. To use this function, load the dashboard sub-API into your page along with the main API:

```
<script src="http://localhost/JsAPI" type="text/javascript"></script>
<script src="http://localhost/JsAPI?api=dash" type="text/javascript"></script>
```

Then call the `loadDashFilters` function:

```
yellowfin.dash.loadDashFilters(dashUUID, callback, arg);
```

The first argument is the unique identifier for the dashboard. The second is a callback function that will be called by the API when the filters for the dashboard have been loaded. The first argument to the callback function will be the list of filters in the dashboard. The second argument to the callback function will be the third argument supplied to the `loadReportFilters` function (if specified).

The filters object returned as the first argument to the callback function is an array containing any analytical filters used in the dashboard, as well as filter group separators. Each element in the array is an object containing information about that filter or filter group. These objects contain the properties:

Properties	Description
`key`	A unique key for this filter or filter group.
`type`	Set to `FILTERGROUP` if this object represents a filter group. Other values indicate a type of analytic filter.
`description`	The description of the filter or filter group.
`groupId`	For filter groups: a numeric identifier for the group.
`state`	For filter groups: set to `OPEN` if the group is currently opened.
`display`	For filters: the display style used by the filter.
`dependencies`	For filters: set to `true` if other filters in the dashboard are dependent on this one.
`list`	For filters: set to `true` if the filter is a list style (allows multiple values).
`between`	For filters: set to `true` if the filter is a between style (requires a start and end value).
`listValues`	For filters: if the filter is displayed as a drop-down list, this property contains a list of available options.

Examples

This example loads the dashboard filters and displays them to the user:

```
function filterCallback(filters) {

   for (var i = 0; i < filters.length; i++) {
      alert('Filter ' + filters[i].description + ' (' +
         filters[i].key + '), display style: ' +
         filters[i].display);
   }

}

yellowfin.reports.loadReportFilters(1234, filterCallback);
```

This function can be used to load the available filters, and then pass them back to the `loadDash` function to set up initial filter values for the dashboard when it is loaded into the page:

```
function filterCallback(filters) {

   var filterValues = {};

   for (var i = 0; i < filters.length; i++) {

      if (filters[i].description == 'Country') {

         filterValues[filters[i].key] = 'Australia';

      } else if (filters[i].description == 'Start Date') {

         filterValues[filters[i].key] = '2011-01-01';

      } else if (filters[i].description == 'Invoiced Amount') {

         filterValues[filters[i].key] = 6400;

      }

   }

   // set up other options to load the dashboard
   var options = {};
   options.dashUUID = '3b0b6c9a-9dfb-41f0-b85a-eb17bb8aeeb9';
   options.elementId = 'myDash';
   // add the filter values
   options.filters = filterValues;

   // load the dashboard
   yellowfin.loadDash(options);

}

yellowfin.dash.loadDashFilters('3b0b6c9a-9dfb-41f0-b85a-eb17bb8aeeb9', filterCallback);
```

Filter values passed to the `loadDash` function should be specified as simple values as above. If the filter is a list style, multiple values can be set using an array:

```
filterValues[key] = ['Australia', 'China', 'Italy'];
```

If the filter is a between style, the start and end values should be set using an array:

```
filterValues[key] = [500, 600];
```

The `options.filters` element passed to the `loadDash` function should contain values keyed by the `keys` returned from the `loadDashFilters` function.

top