LoaderEvents API

LoaderEvents Overview

In Yellowfin 9.4 we enabled the option to add customized loaders to the Javascript API through the “loaderEvents” API.

These loaders are also available in code mode and when using custom headers via the window object.

This allows a developer to listen for instances where an element within a Yellowfin element has a loader added to it or removed from it and then reacts to that event.

There are a number of different actions that could be taken. They include:

- prevent the Yellowfin loader from being added, and add a custom loader onto the same element;
- attach a loader to a different, related element; and.
- log the loader being added or removed (useful during development).

For example, a business using a white-label version of Yellowfin software might prefer to use their own loader icon to match their brand rather than the default blue circle that ships with Yellowfin.

Note: due to the nature of the LoaderEvents API, it has no properties nor events associated specifically with it.

Function Reference

yellowfin.loaderEvents

After the Yellowfin API `init()` call has completed, the `yellowfin.loaderEvents` function will become available. This provides two basic functions — `listen` and `stopListening` — which allow you to set up and remove loader events as you require. It also provides two functions that simplify replacing many child loaders at once — `interceptChildLoadersForElement` and `stopListeningToElementIntercept`.

Note: to use this function in code mode or with custom headers, switch out "yellowfin.loaderEvents" for "window.loaderEvents"

listen(listenObject)

This creates a loader event listener with the passed `listenObject` object.

The `listenObject` object should contain the following parameters:

Parameter	Type	Description	Extra info
`selector`	String	The CSS selector for the object you wish to create the listener on. If you wish to intercept all loaders you can use the selector `'html *'`.	Either `selector` or `element` must be defined to avoid an error that prevents the event listener from being created. Of the two, defining `selector` is preferable.
`element`	HTML Element	The HTML element to listen for loaders being added to. This will only trigger events if the element added to the loader to is the same as the element defined here.	Optional, as an alternative to defining `selector`.
`onAdd`	Function	The function that is called immediately before Yellowfin adds a loader to an element that matches the selector. A `LoaderEvent` will be passed to this callback function.	Either `onAdd` or `onRemove` must be defined. Both can be defined, if needed. If neither are defined, an error will be thrown and the event listener will not be created.
`onRemove`	Function	A callback function which is immediately before Yellowfin code removes a loader from an element that matches the selector. A `LoaderEvent` will be passed to this callback function.	Either `onAdd` or `onRemove` must be defined. Both can be defined, if needed. If neither are defined, an error will be thrown and the event listener will not be created.

stopListening(listenObject)

Removes a loader event, the passed `listenObject` must be the exact same object that was passed into the listen function. Otherwise the listener will not be removed.

interceptChildLoadersForElement(element, listenObject)

When this function is called, Yellowfin intercepts any elements with child loaders and applies those loaders to the passed element rather than the child. If `interceptChildLoadersForElement` is called multiple times on the same element, only the first application will be used.

The `listenObject` parameter for interceptChildLoadersForElement works in a similar manner to the `listenObject` parameter for `loaderEvents.listen`. The `onAdd` function will be called the first time a loader is added to the passed element, and `onRemove` will be called when the loader is removed from the passed element

In the sample code below, the code looks for loaders added anywhere on the page and adds them to the body tag instead.

```
yellowfin.loaderEvents.interceptChildLoadersForElement(document.querySelector('body'), {
    onAdd: function(e) { console.log('loader added to body') },
    onRemove: function(e) { console.log('loader removed from body') }
});
```

stoplisteningToElementIntercept (listenObject)

The `interceptChildLoadersForElement()` listener can be removed by using the `stopListeningToElementIntercept()` function .

Pass only the element you wish to stop listening to.

```
LoaderEvents.stopListeningToElementIntercept(document.querySelector('body'))
```

LoaderEvent

Each time an `onAdd` or `onRemove` function is called, a `LoaderEvent` object will be created. This contains the element the loader is going to be added to, as well as some functions which allow manipulation of how the event is handled.

LoaderEvent.element

This is the element which the loader is going to be added to or removed from.

LoaderEvent.preventDefault()

This event prevents the loader from being added to or removed from the element.

For example, you could initialize your event listener like this:

```
yellowfin.loaderEvents.listen({
    selector: ‘html *’,
    onAdd: function(event) {
        event.preventDefault();
    }
});
```

This would cause no loaders to be added to any Yellowfin elements. This by itself isn’t particularly useful, however you could combine the preventDefault with code that adds your own customized loader.

If you were to create an event listener that only contained an `onRemove` function that prevented default, like this:

```
yellowfin.loaderEvents.listen({
    selector: ‘html *’,
    onRemove: function(event) {
        event.preventDefault();
    }
});
```

This would prevent the loader from being removed from the element. As such you should generally only use `preventDefault` on removal when you are handling the loader removal yourself in the `onRemove` function.

LoaderEvent.setOverrideElement(element)

Defines an element to append the standard Yellowfin loader to, rather than the element that has been passed in the `LoaderEvent` object.

This means that if you wish the standard Yellowfin loader to be on a parent element it can be. When using this function, you must define the `setOverideElement` in both the *onAdd* and *onRemove* functions.

```
yellowfin.loaderEvents.listen({
    selector: 'report-output > div', //Listen for loaders on immediate children of all reports
    onAdd: function(event) {
        event.setOverrideElement(event.element.parentNode); //Adds a loader to the parent node of the passed
element
    },
    onRemove: function(event) {
        event.setOverrideElement(event.element.parentNode); //Removes the loader from the parent node
    }
});
```

top